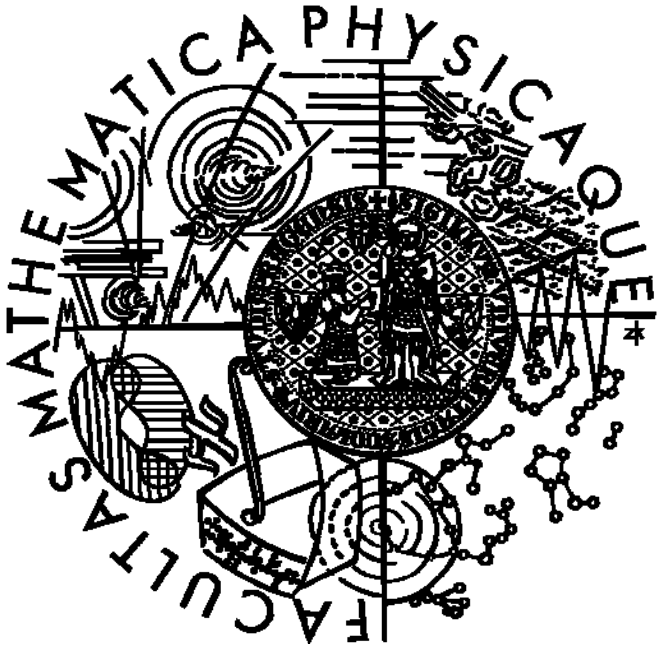
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Vladislav Kozák

Extrakce znalostí z dat

Katedra teoretické informatiky a matematické logiky

Vedoucí práce: **doc. RNDr. Iveta Mrázová, CSc.**

Studijní program: **Databázové systémy**

2009

Ďakujem pani docentke Mrázovej za ochotu, vynaložený čas, energiu a za odborné vedenie diplomovej práce.

Prehlasujem, že som diplomovú prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím zo zapožičaním práce.

V Prahe dňa 30. 7. 2009 Vladislav Kozák

Obsah

Názov práce: Extrakce znalostí z dat
Autor: Vladislav Kozák
Katedra: Katedra teoretické informatiky a matematické logiky
Vedúci diplomovej práce: doc. RNDr. Iveta Mrázová, CSc.
e-mail vedúceho: iveta.mrazova@mff.cuni.cz

Abstrakt: Predmetom diplomovej práce je zhrnutie celkového procesu dobývania znalostí z dát a popis existujúcich algoritmov na predspracovanie dát a tvorbu modelu. Vlastnosti algoritmov sú navzájom porovnané a výsledky podložené opakovateľnými testami. Získané znalosti sú použité a aplikované na 2 úlohách z praxe. Súčasťou diplomovej práce bola tiež tvorba aplikácie na dobývanie znalostí z dát. Pri vývoji aplikácie sa kládol dôraz na robustnosť, použiteľnosť, intuitívnosť ovládania ako aj široké spektrum implementovaných algoritmov na dobývanie znalostí z dát.

Kľúčové slová: dobývanie znalostí, extrakcia pravidiel

Title: Knowledge Extraction from Data
Author: Vladislav Kozák
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: doc. RNDr. Iveta Mrázová, CSc.
Supervisor's e-mail address: iveta.mrazova@mff.cuni.cz

Abstract: The task of this master thesis is to describe the overall process of data mining and algorithms used for data preparation and data modelling. The qualities of these algorithms are compared and the results are well-founded with repeatable tests. Knowledge gained by this research is applied to 2 real data based tasks. Master thesis includes development of own data mining application. The stress was laid on robustness, intuitive GUI as well as wide spectrum of data mining algorithms implemented.

Keywords: data mining, rules extraction

1. Úvod

1.1. Motivácia

Dobývanie znalostí je pomerne mladé odvetvie informatiky, ktoré je úzko spojené s databázovými systémami a matematickou štatistikou. Vyvinulo sa v polovici 90. rokov a v súčasnosti sa jeho možnosti ďalej rozširujú.

Táto vedná disciplína sa zaoberá hľadaním užitočných avšak skrytých informácií, závislostí a záverov v dátach. Objem dát v každej organizácii značne narastá a manažéri sú ochotní platiť nemalé čiastky za technológie sprostredkujúce informácie, ktoré im umožnia lepšie hospodárenie, fungovanie a podporu pre rozhodovanie. Jednoducho povedané: „V dátach je bohatstvo a my ho potrebujeme vyťažiť."

Cieľom diplomovej práce je popísanie, porovnanie a aplikácia postupov a techník, ktoré umožňujú analýzou dát nájsť určité závislosti obsiahnuté v dátach. Nájdené závislosti by mali byť formulované pomocou pravidiel. Na vytipovanie vhodných modelov a ich vzájomné porovnanie budú použité umelé dáta. Reálne dáta použijem na overenie získaných poznatkov v praxi. Výsledky budú vhodne znázornené a dosiahnuté závery sa pokúsim odôvodniť. Proces dobývania znalostí silne závisí na kvalite a povahe dát, preto je garancia výsledkov neistá.

V prvej časti práce popíšem teoretické znalosti a algoritmy, ktoré sa následne pokúsim aplikovať na umelé dáta a 2 reálne zdroje dát. Popísané budú neurónové siete, viacero algoritmov na generovanie rozhodovacích stromov, metóda k-najbližších susedov, metóda analýzy nákupného košíka a vlastná modifikácia tohto algoritmu pre účely klasifikácie.

Pri spracovávaní a analýze dát sa použijú nasledujúce databázové systémy a programovacie prostredia:

- *Microsoft SQL Server 2005 Express* – databázový systém od spoločnosti Microsoft, ktorý je vo verzii Express voľne dostupný.
- *Oracle 10g* – moderný multiplatformový databázový systém s veľmi pokročilými možnosťami spracovania dát a vysokým výkonom.
- *R* – je voľne šíriteľný softvér používaný na štatistické výpočty a vizualizáciu.

Dôvodom voľby uvedených databázových systémov je fakt, že dáta sú vo forme exportov z databáz práve týchto databázových systémov. Štatistický softvér R som vybral hlavne pre jeho jednoduchosť a ľahké ovládanie.

V prvých troch kapitolách zhrniem teoretické základy potrebné na porozumenie reálnych úloh dobývania znalostí z praxe. Popíšem kľúčové postupy a algoritmy, ktoré miestami vyžadujú malé modifikácie, aby sa viac hodili na riešenie konkrétnej úlohy. Nasledujú 2 kapitoly, z ktorých každá popisuje proces dobývania znalostí na reálnych dátach. V kapitole 7 s názvom

„Vytvorená aplikácia“ popíšem postup tvorby a užívateľskú dokumentáciu k aplikácii, ktorú som vytvoril za účelom demonštrácie výsledkov.

Aplikácia je vytvorená vo vývojom prostredí *Microsoft Visual Studio C# 2008 Express Edition*. Väčšinu analyzovaných algoritmov som sa rozhodol implementovať ako databázové procedúry. Dôvody tohto rozhodnutia popisujem v kapitole 7.1. Môj predpoklad, že takáto implementácia bude rýchlejšia oproti klasickej aplikačnej implementácii, sa pokúsim overiť v závere kapitoly 4.6 venovanej testovaniu algoritmov.

Väčšina postupov a algoritmov už je v nástrojoch na spracovanie dát implementovaná. Ich použitiu sa však pokúsim vyhnúť. Jeden dôvod je, že pre účely diplomovej práce bude lepšie, keď bude proces transparentný, pričom implementované algoritmy v daných nástrojoch môžu vytvárať dojem akejsi čiernej skrinky. Druhým dôvodom je fakt, že pre žiadny z popisovaných algoritmov sa mi nepodarila nájsť hotová a funkčná implementácia vo forme databázovej procedúry.

2. Metodológie

Nikto nevie zaručene povedať, aký postup pri dobývaní znalostí nám v konkrétnom prípade zabezpečí najlepšie výsledky. Použitím skúseností z úspešných projektov sa však časom vytvorilo niekoľko súborov návodov, ktoré určujú poradie a náplň jednotlivých krokov s cieľom maximalizovať šance na získanie použiteľných výsledkov.

Niektoré z nich sú produktom výskumu a vznikli na akademickej pôde (CRISP-DM), iné sú výsledkom praktických skúseností producentov softwaru na dobývanie znalostí z dát (5A, SEMMA).

Napriek rozličnému pomenovaniu sú jednotlivé kroky a operácie, ktoré podľa týchto metodológií treba vykonať, značne analogické. Preto pri metodológiách 5A a SEMMA vyznačím, ktorým krokom v metodológii CRISP-DM odpovedá daný krok. Obsah a význam týchto krokov je detailne popísaný v kapitole 3.

2.1. CRISP-DM

Názov metodológie je odvodený z anglického slovného spojenia *CRoss Industry Standard Process for Data Mining*, ktorý sa voľne prekladá ako Štandardný model procesu pre dobývanie znalostí z dát. Vývoj tejto metodológie bol iniciovaný Európskou komisiou. Cieľom bolo vytvoriť súbor postupov, ktoré by poskytovali štandardizovaný návod na riešenie rozličných úloh spojených s dobývaním znalostí. Za hlavné výhody tejto metodológie sa považujú nezávislosť na konkrétnom odvetví a použitých nástrojoch a softvéroch, príbuznosť s procesným modelom KDD a celkové ukotvenie procesu dobývania znalostí z dát.

Výskum

Výskum bol vedený konzorciom CRISP-DM, ktorý tvoria 4 gigantické subjekty: ISL, NCR, Daimler Crysler and OHRA. Vlastníkom metodológie je práve toto konzorcium. Vývoj prvej verzie 1.0 bol dokončený v roku 1999. Bolo však jasné, že metodológia bude vyžadovať revíziu a v júli 2006 ohlásilo konzorcium zahájenie výskumu novej verzie.

Základom výskumu metodológie CRISP-DM boli početné skúsenosti z úspešných projektov. Tento univerzálny návod preto dáva najlepšie predpoklady k tomu, aby jeho použitie viedlo k zvýšeniu rýchlosti, ekonomickosti, spoľahlivosti a efektívnosti pri hľadaní znalostí.

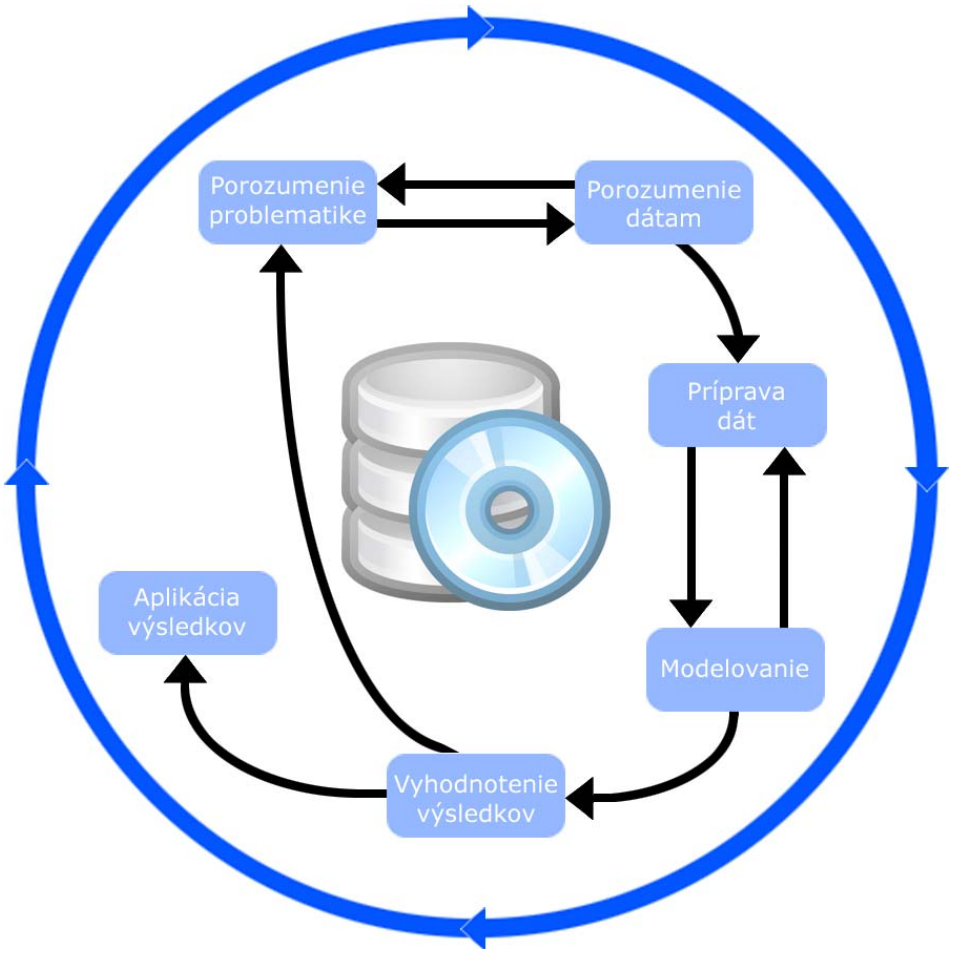
Popis metodológie

CRISP-DM rozdeľuje proces dobývania znalostí do 6 hlavných krokov:

- Porozumenie problematike (*Business understanding*)
- Porozumenie dátam (*Data understanding*)
- Príprava dát (*Data preparation*)

- Modelovanie (*Modeling*)
- Vyhodnotenie výsledkov (*Evaluation*)
- Aplikácia výsledkov (*Deployment*)

Dobývanie znalostí je touto metodológiou definované ako iteratívny proces. Znamená to, že k niektorým krokom sa opätovne vraciame a prehodnocujeme závery tohto kroku z predošlej iterácie, upravujeme ich a tieto úpravy distribuujeme ďalej. Najlepšiu predstavu získame pohľadom na obrázok, ktorý reprezentuje procesný diagram metodológie CRISP-DM:



obr. 1 – Procesný diagram metodológie CRISP-DM

Z obrázku je možné vyčítať, ktoré prechody medzi jednotlivými krokmi sú očakávané a potrebné. Vidíme, že výsledok v jednom kroku ovplyvňuje voľbu nasledujúceho kroku. Hlavný kruh dookola vyjadruje cyklickú povahu procesu, definovanú touto metodológiou. Kroky vyžadujú špecifický dôraz a majú aj rozličné časové nároky. Porozumenie problému je považovaný za najdôležitejší hlavne z hľadiska úspešnosti projektu. Z časového hľadiska je najťažšou úlohou príprava dát, ktorá zaberá 80% celkového času. Obecne je však veľmi náročné odhadnúť, koľko absolútneho času by nám mala celá analýza zabrať.

2.2. SEMMA

SEMMA je výsledkom spoločnosti SAS v snahe vytvoriť vlastnú robustnú metodológiu. Skratka SEMMA vyjadruje hlavné kroky procesu dobývania znalostí z dát:

- *Sample* – vybranie vhodných objektov
- *Explore* – vizualizácia, porozumenie dátam, redukcia dát
- *Modify* – transformácia a modifikácia dát, príprava dát na modelovanie
- *Model* – analýza dát, aplikovanie vybranej metódy/algoritmu na dobývanie znalostí
- *Assess* – vyhodnotenie modelu, interpretácia, aplikácia výsledkov

Metodológia pozostáva z 5 krokov, čo je o jeden menej, ako definuje metodológia CRISP-DM. Vidieť však, že náplň krokov ako aj postup je takmer zhodný. Jediným rozdielom je, že posledný krok „*Assess*“ zahrňuje posledné 2 kroky metodológie CRISP-DM.

2.3. 5A

Metodológia 5A je vyvinutá spoločnosťou SPSS. Je výsledkom dlhoročných skúseností tejto spoločnosti v oblasti dobývania znalostí. Metodológia definuje proces dobývania znalostí a delí ho na 5 hlavných krokov:

- *Assess* – definovanie úlohy a cieľa projektu
- *Access* – zber, prístup k dátam a ich príprava na analýzu
- *Analyze* – samotná analýza
- *Act* – premena znalostí na akčné znalosti
- *Automate* – aplikácia výsledkov a nadobudnutých znalostí

Štvrtý krok „*Act*“ sa zaoberá zberom a interpretáciou výsledkov. Ako píše [5], výsledky by mali byť v tomto kroku prekladané do jasnej a zrozumiteľnej podoby.

Podobne ako v prípade metodológie SEMMA, aj tu vidieť analógiu krokov s krokmi metodológie CRISP-DM. Metodológia 5A má 5 krokov, pričom druhý krok „*Access*“ zahrňuje operácie, ktoré metodológia CRISP-DM definuje v krokoch 2 a 3 – „Porozumenie dátam“ a „Príprava dát“. Poradie aj obsah ostatných krokov sú zhodné.

3. Kroky procesu analýzy

V tejto kapitole popíšem význam a obsah jednotlivých krokov procesu dobývania znalostí. Názvy krokov sú odvodené z terminológie metodológie CRISP-DM. V predchádzajúcej kapitole sme si však ukázali, že napriek rozdielnemu pomenovaniu, sú poradie aj náplň krokov zhodné s metodológiami SEMMA a 5A.

3.1. Porozumenie problematike a určenie cieľa

Porozumenie problematike

Základným krokom v snahe uspieť pri dobývaní znalostí z dát je orientácia v problematike. Ak neporozumieme problematike, nie sme schopný na adekvátnej úrovni komunikovať s našim zákazníkom a porozumieť, čo si od našej analýzy sľubuje. Tento krok býva často podceňovaný. Ako tvrdí [1], ani hlboké znalosti o algoritmoch a postupoch na dobývanie znalostí z dát nám nezaručia úspech, pokiaľ sa nezorientujeme v danom odbore.

Stanovenie cieľa

Stanovenie cieľa je vo všetkých publikáciách a metodológiách, ktoré som študoval, označovaný ako najpodstatnejší krok celého dobývania znalostí. Toto tvrdenie je podložené praktickými skúsenosťami autorov.

Na začiatku je však ťažké stanoviť ciele, pokiaľ nevieme nič o obsahu, kvalite a povahe dát, ktoré boli na analýzu poskytnuté. Na strane druhej je prakticky nemožné analyzovať dáta, ak nevieme, čo požadujeme. Viaceré metodológie preto deklarujú, že proces stanovenia cieľu je iteratívny. Na začiatku stanovujeme relatívne málo konkrétny cieľ (s dostatočným nadhľadom), ku ktorému sa neskôr vraciame a upresňujeme ho.

V neposlednom rade je dôležité formulovať ciele z pohľadu manažéra. Typickými príkladmi cieľov v tejto fáze môžu byť: získanie nových zákazníkov, identifikácia neúspešného produktu pred jeho zavedením na trh alebo zvýšenie zisku z predaja použitím akciových ponúk.

Kritéria hodnotenia a zdroje

Na záver tohto kroku je dôležité si vytvoriť plán, podľa ktorého budeme schopný dosiahnuté výsledky otestovať a rozhodnúť, či boli naplnené stanovené ciele. Na toto testovanie môžeme použiť konkrétny výpočet alebo algoritmus, ktorého výsledky sú merateľné. Často, a to hlavne pri malých projektoch, sa používa subjektívne ohodnotenie osobou zo strany zadávateľa, ktorá problematike dostatočne rozumie.

Ďalej je potrebné zvážiť, aké zdroje budú poskytnuté:

- Čas – počet človeko-hodín, ktorý bol podľa predpokladov na vyriešenie úlohy vyhradený

- Financie – koľko finančných zdrojov máme na projekt dostupných
- Hmotné prostriedky – dostupný hardvér, priestory
- Softvér – aké programové vybavenie máme k dispozícii
- Ľudské zdroje – odborníci v súvisiacich odboroch

Podľa skúseností autora publikácie [3] žiadnu z uvedených vecí netreba podceniť a i drobný prehrešok môže mať za následok neúspech celého projektu.

3.2. Porozumenie dátam

Keď už máme predstavu, aké znalosti hľadáme, pristupujeme k zdroju dát a zoznamujeme sa s tým, aké dáta a v akom formáte sú k dispozícii. Posudzujeme tiež kvalitu dát a snažíme sa vytipovať podmnožiny dát, ktoré by pre nás mohli byť zaujímavé. Tento krok by som rozdelil na 2 časti.

Technická príprava

Dáta od zadávateľa môžeme dostať v rozličnej podobe a zdroj dát často vyžaduje nejakú konverziu, import alebo iné technické kroky, ktoré musíme podniknúť, aby sme si dáta mohli čo i len prehliadnuť.

Medzi najznámejšie zdroje dát, ktoré sa používajú, patria export (*dump*) z niektorého z databázových systémov, súbor vo formáte *ASCII*, alebo súbor vo formáte niektorého z tabuľkových procesorov[1] (xls,). V prípade exportu z databázy musíme identifikovať, o ktorý databázový systém sa jedná a nájsť spôsob, ako dáta presunúť do nami zvoleného databázového systému, prípadne dáta previesť do iného zvoleného formátu.

Súbor vo formáte ASCII, tiež prezývaný *flat file*, je súbor obsahujúci dáta v „čitateľnej forme“. Jednotlivé riadky súboru predstavujú záznamy. Atribúty záznamu sú oddelené zvoleným oddeľovačom (separátorom), preto sa pre tieto súbory zaužívala prípona „*.csv*“, kde CSV je skratka *comma-separated values*. Obecne je možné použiť akékoľvek oddeľovače záznamov a ich atribútov. Preto pravidlo „jeden riadok = jeden záznam“ je len dobre zaužívanou konvenciou.

Tabuľkové procesory poskytujú dostatočné nástroje pre porozumenie dátam. Pre neskoršie pracovanie s dátami sa zvyčajne dáta exportujú do CSV súboru.

Prieskum dát

V tejto časti skúmame, aké dáta sú k dispozícii a aká je ich štruktúra. Je dôležité, aby sme spolu s dátami dostali aj dátový slovník, ktorý nám

[1] Tabuľkový procesor (anglicky *spreadsheet*) je program spracovávajúci tabuľku informácií. Patria k najstaršiemu softvérovému vybaveniu osobného počítača. Medzi najznámejšie patria Microsoft Excel, OpenOffice.org Calc, Quattro Pro alebo Lotus.

prezradí názov, význam, dátový typ, interval prípadne doménu daných atribútov. Tieto informácie sa nazývajú tiež *metadáta*. Neprítomnosť týchto pomocných dát býva často problém a odvodzovať tieto informácie z hodnôt je časovo veľmi náročný a nie vždy úspešný proces.

V databázových systémoch sme v prípade neexistujúceho popisu dát schopný vyčítať aspoň dátový typ atribútu a cez cudzie kľúče aj vzťahy jednotlivých entít. V prípade, že sa však použijú nič nehovoriace skratky pre názvy tabuliek a ich atribútov, je interpretovanie týchto dát takmer nemožné.

Keď už máme potrebné informácie o dátach, musíme skontrolovať, či máme dáta nevyhnutné na naplnenie stanovených cieľov. Je logické, že nemôžeme vytvoriť segmentáciu trhu podľa veku, keď v zdroji dát nemáme informáciu o veku, ani inú informáciu (napr. rodné číslo), z ktorej by sme mohli vek odvodiť.

Do tohto kroku by som tiež zaradil elimináciu nepotrebných a redundantných dát. Keď už vieme identifikovať nepotrebné dáta, ako napríklad nesúvisiace tabuľky, je dobré tieto dáta postupne odstraňovať. Zväčší sa tým prehľad a zjednoduší práca s relevantnými dátami.

Keď už máme zhruba predstavu, ktoré dáta by pre nás mohli byť zaujímavé, zhodnotíme kvalitu dát. Zobrazujeme si rôzne štatistické údaje ako priemery, minimá a maximá, mediány, identifikujeme intervaly, domény a frekvencie hodnôt atribútov. Zatiaľ je tento proces dôležitý len na získanie prehľadu o dátach. Podrobnejšie skúmanie a rôzne štatistické a vizualizačné techniky sú popísané v nasledujúcej kapitole.

3.3. Príprava dát

V tomto kroku by sme mali dospieť k vytvoreniu výsledného dátového súboru, na ktorý budeme schopný aplikovať metódy na dobývanie znalostí z dát. Predspracovanie dát je nevyhnutným krokom pre dosiahnutie kvalitných výsledkov.

Tento proces zahrňuje identifikáciu a odstránenie chybných dát, nájdenie extrémov, vyhľadanie chýbajúcich hodnôt a ich doplnenie rozumnými hodnotami. Niektoré dáta musíme transformovať, aby viac vyhovovali potrebám pre modelovanie.

Jednotlivé metódy a algoritmy predspracovania dát som čerpal z publikácie [5]. Doplňujúce informácie som čerpal z publikácií [1] a [3].

Klasifikácia dát

Dáta delíme na 2 hlavné triedy: ***kvalitatívne*** a ***kvantitatívne***. Kvantitatívne dáta majú podobu číselných hodnôt. Kvalitatívne dáta sú reprezentované textovými hodnotami. Dáta z tejto triedy môžu byť prevedené na kvalitatívne tým, že sa zakódujú – každému reťazcu z domény sa priradí číselná hodnota. Toto je obvyklé v databázach, kde sa aj kvôli zachovaniu

3NF[2] obvykle opakujúce textové hodnoty dávajú do osobitných tabuliek, ktoré sa prezývajú číselníky.

Nominálne (kategoriálne) dáta sú číselné dáta, ktoré odpovedajú kategoriálnym dátam. Je dôležité si uvedomiť, že tieto číselné hodnoty nemajú žiadny relatívny význam. To znamená, že keď nejaký atribút A_1 má číselnú hodnotu x a A_2 má číselnú hodnotu 2*x, nemôžeme povedať že A_2 je 2 krát lepší alebo väčší.

Ordinálne (kategoriálne) dáta sú tiež číselné dáta, ktoré odpovedajú kategoriálnym dátam. Na rozdiel od nominálnych ale majú relatívny význam. Číselná hodnota atribútu môže vyjadrovať určitú silu a dôležitosť atribútu oproti ostatným atribútom. Na ukážku si predstavme, že máme databázu, ktorá obsahuje dáta insolventný register zákazníkov. Každému zákazníkovi priradíme hodnotenie od 1 do 10, ktoré vyjadruje mieru zadlženosti subjektu. Keď vezmeme subjekt A s hodnotením 3 a subjekt B s hodnotením 7, vieme podľa hodnôt porovnať, ktorý subjekt predstavuje väčšie riziko.

Intervalové dáta sú číselné dáta, u ktorých má význam, na rozdiel od ordinálnych dát, používať na vyjadrenie vzťahu aj operácie ako sčítanie a odčítanie. V tom prípade sa používa väčší interval, ktorý umožňuje jemnejšie hodnotenie.

Posledným typom sú ***spojité dáta***, ktoré sú vhodné na používanie všetkých aritmetickým operácií. Väčšina produkčných dát je práve tohto typu. Typickým príkladom sú platby v obchode.

Vzorkovanie

Vzorkovanie je metóda, ktorá sa používa na zrýchlenie procesu dobývania znalostí. V dobách, keď výkony procesorov, priepustnosti zberníc a rýchlosti aj kapacity pevných diskov boli ďaleko nižšie ako dnes, bola táto metóda na prípravu dát nevyhnutná. Dnes je ale stále dosť používaná, pretože sa pri jej použití dosahujú prakticky rovnaké výsledky ako pri použití úplných dát.

Z dát treba vybrať pre každú hodnotu cieľového atribútu určité percento zástupcov. Váhu pre danú hodnotu dostaneme ako pomer medzi počtom všetkých záznamov s touto hodnotou cieľového atribútu a vybraných záznamov. Po tejto operácii máme výslednú množinu dát pre modelovanie, ktorú pred samotným procesom musíme ešte vyčistiť a transformovať. Čistenie dát sa skladá z identifikácie odľahlých hodnôt, doplnenia chýbajúcich dát a identifikácie a opravy chybných dát.

Eliminácia atribútov

Častá je situácia, kedy dáta na analýzu obsahujú veľké množstvo (až stovky) atribútov, pričom veľa z nich je nepoužiteľných. Dôvodom k tomu môže byť fakt, že atribút je irelevantný (napr. obsahuje náhodné dáta) alebo

[2] Tretia normálna forma (3NF) je metodika (súbor doporučení) pre návrh dátového modelu databáze, ktorého použitie zlepšuje integritu dát, konzistenciu dát a rýchlejšie prevedenie operácii v transakčných databázach.

korelovaný[3]. V takom prípade nám dobrá znalosť dát a porozumenie problematike pomôže identifikovať a eliminovať takéto atribúty.

V opačnom prípade je možné použiť automatické metódy, ktoré sa delia na 2 skupiny:

- Transformácia
- Selekcia

Pri ***transformácii*** vytvárame nové atribúty kombináciou existujúcich. Z niekoľkých atribútov $a_1,...,a_n$ tak môže vzniknúť atribút a, ktorý je lineárnou kombináciou týchto atribútov a lepšie vyhovuje potrebám modelovania. Príkladom takýchto automatických metód sú faktorová analýza alebo Karhunenov-Loevov rozvoj. Nevýhodou tohto prístupu je, že atribúty musia byť numerické. Okrem toho z transformovaného atribútu nie sme schopný spočítať pôvodné hodnoty atribútov, čím strácame možnosť zmysluplnej interpretácie výsledkov.

Selekcia je proces, pri ktorom sa hľadajú atribúty, na ktorých je klasifikácia najviac závislá. Publikácia [5] identifikuje na základe spôsobu prístupu k tomuto problému 2 metódy prístupu. Prvá z nich je *metóda filtru (filter approach)* a spočíva v tom, že sa pre každý atribút spočíta podľa zvolenej metódy charakteristika, ktorá určuje mieru prispievania tohto atribútu ku klasifikácii. Túto vlastnosť meriame pomocou vzájomnej informácie, hodnoty X^2 (chí-kvadrát), entrópie alebo informačného zisku. Informačný zisk a entrópia sú detailne popísané v kapitole o rozhodovacích stromoch.

Druhou metódou prístupu je *metóda obálky (wrapper approach)*. Pri tejto forme prístupu k problému sa používa niektorý z modelovacích algoritmov a na podmnožine atribútov sa vytvorí model. Následne porovnaním skóre/kvality týchto modelov pre jednotlivé podmnožiny atribútov sa určí tá najvhodnejšia podmnožina. Takýto prístup môže byť extrémne pomalý, pretože pri n vstupných atribútoch existuje 2^n-1 možných kombinácií. Použitím vzorkovania dokážeme proces tvorby modelu značne urýchliť. S narastajúcou výpočtovou silou sa preto tento prístup dostáva čoraz viac do popredia.

Hľadanie odľahlých hodnôt

Metóda hľadania odľahlých hodnôt sa používa na identifikáciu chýb v dátach. Odľahlá hodnota (*outlier*) je taká hodnota niektorého z atribútov, ktorá sa vyskytuje s veľmi nízkou frekvenciou ďaleko od strednej hodnoty a od ostatných hodnôt. Na automatickú identifikáciu odľahlých hodnôt existuje viacero metód. Medzi najznámejšie patrí *Chauvenetovo kritérium*, *Grubbov test* a *Peirceovo kritérium*.

Grubbov test predpokladá normálne rozdelenie a nedáva dobré výsledky pri malom počte hodnôt. Pre menej ako 6 hodnôt je prakticky nepoužiteľný. Grubbov test identifikuje a vylúči z dát v každom behu jednu odľahlú hodnotu. Proces sa opakuje, kým nie je identifikovaná žiadna odľahlá

[3] V štatistike korelácia vyjadruje vzájomný lineárny vzťah medzi 2 veličinami, v našom prípade atribútmi.

hodnota. Nulovou hypotézou je neexistencia odľahlej hodnoty, proti čomu stojí alternatívna hypotéza o existencii aspoň jednej takejto hodnoty. Nech atribút *A* nadobúda *N* hodnôt, ktoré sú označené $a_1,...,a_N$. Grubbova štatistika je definovaná ako

$$G = \frac{\max_{i=1,..,N} |a_i - \mu|}{s},$$

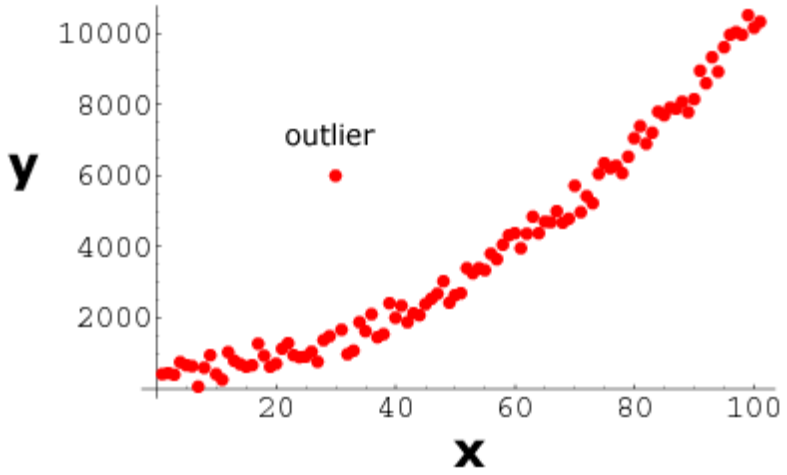
kde *s* označuje smerodajnú odchýlku a μ je stredná hodnota, ktorá v prípade normálneho rozdelenia odpovedá priemeru. Pre obojstranný Grubbov test zamietame na hladine α nulovú hypotézu o neexistencii odľahlých hodnôt ak

$$G > \frac{N-1}{\sqrt{N}} \sqrt{\frac{t^2_{\alpha/2N,N-2}}{N-2+t^2_{\alpha/2N,N-2}}}$$

kde $t^2_{\alpha/2N,N-2}$ označuje hornú kritickú hodnotu t-rozdelenia s *N-2* stupňami voľnosti.

Článok [7] uvádza jednoduchú štatistickú metódu ktorá je založená na Čebyševovom teoréme. Za odľahlú hodnotu označíme takú hodnotu *a* atribútu A_i, ktorá nespadá do intervalu $[\mu_i-\varepsilon\sigma_i,\mu_i+\varepsilon\sigma_i]$, kde μ_i označuje strednú hodnotu a σ_i označuje smerodajnú odchýlku atribútu A_i. Experimentálnou metódou autori článku overili, že najlepšie výsledky dáva z množiny {3,4,5,6} voľba parametru $\varepsilon = 5$.

Znalosť danej problematiky a dát by nám však mala stačiť. Identifikácia nezdôvodniteľných hodnôt pri vizuálnom znázornení je za týchto okolností často pomerne ľahká úloha. Na obr. 2 vidíme typický príklad odľahlej hodnoty. Ani jedna z popísaných automatických metód by však túto hodnotu neidentifikovala. To poukazuje na veľký význam vizualizácie dát.



obr. 2 – Odľahlá hodnota

Chýbajúce dáta

Prvé riešenie, ktoré neznalého dobyvateľa znalostí napadne, je záznamy s chýbajúcou hodnotou vyhodiť. Za akceptovateľné by som považoval, keby sme pri vzorkovaní vyberali len z úplných záznamov. Aj tak ale riskujeme, že konečné závery môžu byť za určitých predpokladov skreslené ba až nepravdivé. Nehovoriac o tom, že publikácia [1] uvádza príklad, kde by nám táto taktika zatvorila dvere k nájdeniu kľúčovej informácie.

Uvádza sa tu istá firma, ktorá prepojila zoznam svojich zákazníkov s externým zdrojom, ktorý obsahoval zoznam predošlých zásielok na oslovenie týchto zákazníkov. Firma chcela identifikovať, ktorým zákazníkom má rozposlať ponuky na nové produkty a hľadala závislosť aj na predošlých ponukách. Niektorý zákazníci však v externom zdroji nemali záznamy a teda im chýbala informácia o počte predošlých zásielok, poslednej zásielke a podobne. Nakoniec sa ukázalo sa, že medzi respondentmi na túto ponuku bolo najviac práve zákazníkov, ktorý nedostali žiadne predošlé zásielky. Keďže ponúk dostávali málo, venovali im väčšiu pozornosť.

Druhou možnosťou je rozhodnutie nahradiť chýbajúcu hodnotu niečím zmysluplným. Hlavným cieľom je doplniť čo najpravdepodobnejšiu hodnotu a zachovať celkové rozdelenie hodnôt atribútu. Metódy na doplnenie chýbajúcej hodnoty sa líšia pre spojité a kategoriálne atribúty. Nižšie si predstavíme hlavnú myšlienku niektorých z nich.

V prípade spojitého atribútu môžeme použiť na substitúciu strednú hodnotu atribútu. V prípade, že je rozdelenie tohto atribútu veľmi nesúmerné, poslúži ako lepší kandidát medián alebo mód. Efektívnejšie je však ***doplnenie strednej hodnoty triedy***, pri ktorom sa dáta rozdeľujú do podskupín podľa iných atribútov. Musíme teda nájsť kategoriálny atribút, ktorý najviac koreluje s atribútom, pre ktorý doplňujeme chýbajúcu hodnotu. Ako hodnotu potom doplníme strednú hodnotu atribútu v tej istej triede, akú má záznam, ktorého

hodnotu dopĺňame. Je potrebné si ale uvedomiť, že táto metóda môže byť za určitých okolností nevhodná a produkovať neočakávané výsledky. Ak napríklad skupina dát A tvorí podľa dopĺňaného atribútu 2 zhluky, ktoré sú vzdialené a medzi nimi ležia dáta zo skupiny B, môže mať dopĺňanie strednej hodnoty triedy úplne opačný efekt.

Poslednou metódou, ktorú pre spojité atribúty spomeniem je ***regresná substitúcia***. Je podobná ako metóda doplnenia strednej hodnoty triedy, ale je aplikovateľná aj na spojité atribúty.

V prípade kategoriálnych atribútov sa odporúča vytvoriť pre chýbajúce hodnoty novú kategóriu. To nám umožní skúmať, či chýbajúca hodnota nemá pre nás nejaký špeciálny význam.

Zaujímavé riešenie doplnenia chýbajúcich hodnôt uvádza publikácia [5]. Kľúčom je použitie niektorého z klasifikačných modelovacích algoritmov, pričom atribút s chýbajúcimi hodnotami sa použije ako cieľový atribút. Chýbajúce hodnoty sa doplnia použitím tohto modelu.

Ošetrenie chybných dát

Pre spojité atribúty je najlepšie použiť na identifikáciu chybných dát hľadanie odľahlých hodnôt, ktoré už bolo popísané. V prípade kategoriálnych atribútov sa kontrolujú frekvencie jednotlivých hodnôt z domény atribútu. Kandidátom na chybné hodnoty sú hodnoty s extrémne nízkymi frekvenciami, väčšinou 1. Po identifikácii chýb máme viacero volieb, ako hodnotu napraviť – najčastejšie sa použije modus (najpočetnejšia hodnota) alebo modus triedy, ktorý je analogický s dopĺňaním strednej hodnoty triedy u chýbajúcich hodnôt spojitých atribútov.

Diskretizácia

Časť modelovacích algoritmov predpokladá, že vstupné atribúty sú kategoriálne. V takom prípade je potrebné spojité atribúty kategorizovať. Pri tomto procese sa interval hodnôt spojitého atribútu rozdelí na niekoľko navzájom disjunktných intervalov a každému intervalu sa priradí jedna z kategórií. Vytvorí sa nový atribút a každému záznamu sa pre tento atribút zvolí kategória príslušného intervalu, do ktorého podľa hodnoty pôvodného spojitého atribútu náleží. Je dôležité si uvedomiť, že tá istá kategória môže byť priradená viac ako jednému intervalu.

Intervaly pre jednotlivé kategórie je možné identifikovať automaticky, alebo využiť osobu (napr. zo strany zákazníka), ktorá má potrebné znalosti a orientáciu v problematike. Automatická diskretizácia môže brať ohľad na hodnotu cieľového atribútu, alebo len rozdeliť interval hodnôt na fixný počet ekvidistantných[4] resp. ekvifrekventných[5] intervalov. Z algoritmov automatickej

[4] Ekvidistantné intervaly sú intervaly rovnakej dĺžky. Podľa požadovaného počtu intervalov, minimálnej a maximálnej hodnoty iba rozdelíme pôvodný interval hodnôt.

[5] Ekvifrekventné intervaly sú intervaly s rovnakým počtom hodnôt. Na rozdiel od ekvidistantných intervalov potrebujeme poznať aj frekvencie jednotlivých hodnôt.

diskretizácie sú podľa [5] najznámejšie algoritmy Fayyadov-Iraniho, Leeho-Shinov algoritmus a algoritmus pre systém KEX.

Fayyadov-Iraniho algoritmus

Autormi algoritmu sú Fayyad a Irani a algoritmus bol publikovaný v roku 1993. Algoritmus postupným delením intervalov podľa jedného deliaceho bodu (binarizácia) rozdelí pôvodný interval na niekoľko menších navzájom disjunktných intervalov. Kritérium pre nájdenie najlepšieho deliaceho bodu a rozhodnutia, kedy skončiť je informačný zisk a entrópia.

1. zotrieď dáta podľa hodnoty zvoleného atribútu vzostupne
2. rekurzívne deľ aktuálny interval *Int* tak, že:
   a. nájdi najlepší deliaci bod *d* a urči informačný zisk $Z(A_{Int,d})$
   b. ak je $Z(A_{Int,d}) > MDL$, tak
      i. rozdeľ interval podľa bodu *d*
      ii. pokračuj v rekurzii

MDL označuje tzv. minimálnu deskriptívnu dĺžku (*Minimal Description Length*) určujúcu, dokedy má zmysel interval deliť. Nech *Int* je interval, *d* je deliaci bod, *k* je počet rôznych tried pre záznamy spadajúce do intervalu *Int*, k_1 nech je počet rôznych tried, ktoré spadajú do intervalu $Int_{<d}$ a k_2 počet rôznych tried, ktoré spadajú do intervalu $Int_{>d}$. Ďalej nech $n(A_{Int})$ označuje počet príkladov, ktoré majú hodnotu atribútu *A* z intervalu *Int*. V prípade $n_t(A_{Int})$ počítame len tie záznamy, ktoré patria do triedy *t*. Potom $Z(A_{Int,d})$ a *MDL* spočítame nasledovne:

$$Z\left(A_{Int,d}\right) = H\left(A_{Int}\right) - H\left(A_d\right)$$

$$MDL = \frac{\log_2\left(n\left(A_{Int}\right)-1\right)}{n\left(A_{Int}\right)} + \frac{\Delta_A\left(Int,d\right)}{n\left(A_{Int}\right)}$$

$$\Delta_A\left(Int,d\right) = \log_2\left(3^k - 2\right) - kH\left(A_{Int}\right) - k_1 H\left(A_{<d}\right) - k_2 H\left(A_{>d}\right)$$

Entrópia $H(A_{Int})$ sa vzťahuje k intervalu pred delením a $H(A_d)$ po delení. Jednotlivé entrópie spočítame nasledovne:

$$H\left(A_{Int}\right) = -\sum_{t=1}^{T} \frac{n_t\left(A_{Int}\right)}{n\left(A_{Int}\right)} \log \frac{n_t\left(A_{Int}\right)}{n\left(A_{Int}\right)}$$

$$H\left(A_d\right) = \frac{n\left(A_{<d}\right)}{n\left(A_{Int}\right)} H\left(A_{<d}\right) + \frac{n\left(A_{>d}\right)}{n\left(A_{Int}\right)} H\left(A_{>d}\right)$$

Informačný zisk a entrópia sú podrobnejšie vysvetlené v kapitole 4.1 na strane 25 s názvom Rozhodovacie stromy.

Algoritmus pre systém KEX

Autorom tohto algoritmu je Petr Berka. Algoritmus vytvára intervaly zhora nadol, pričom počet intervalov nie je dopredu špecifikovaný. Algoritmus sa usiluje o vytvorenie intervalov, v ktorých záznamy patria do tej istej triedy (majú zhodnú hodnotu cieľového atribútu), alebo je dominancia tejto triedy v rámci intervalu výrazná. Pre interval *Int*, atribút *A* a hodnotu *c1* cieľového atribútu *C* chceme, aby sa *P(C=c1|A(Int))* a *P(C=c1)* signifikantne líšilo.

1. ***Hlavný cyklus***
   - 1.1. vytvor usporiadaný zoznam hodnôt uvažovaného atribútu
   - 1.2. pre každú hodnotu
     - 1.2.1. spočítaj frekvenciu tejto hodnoty pre jednotlivé triedy
     - 1.2.2. priraď kód triedy každej hodnote procedúrou ASSIGN
   - 1.3. vytvor intervaly hodnôt procedúrou INTERVAL
2. ***ASSIGN***
   - 2.1. ak pre danú hodnotu patria všetky objekty do rovnakej triedy, tak priraď hodnote kód tejto triedy
   - 2.2. inak
     - 2.2.1. ak pre danú hodnotu sa rozdelenie objektov do tried signifikantne líši (na základe X^2-testu) od apriórneho rozdelenia tried, tak priraď hodnote kód najfrekventovanejšej triedy
     - 2.2.2. inak priraď hodnote kód „?“
3. ***INTERVAL***
   - 3.1. ak sekvencia hodnôt má rovnaký kód triedy, vytvor interval z týchto hodnôt
   - 3.2. pre každý interval INT_i
     - 3.2.1. ak interval INT_i patrí do triedy „?“
       - 3.2.1.1 ak susedné intervaly INT_{i-1}, INT_{i+1} patria do tej istej triedy, tak vytvor interval zjednotením INT_{i-1}, INT_i a INT_{i+1}
       - 3.2.1.2 inak vytvor interval buď spojením INT_i a INT_{i-1} alebo INT_i a INT_{i+1} na základe výsledku X^2-testu
   - 3.3. vytvor spojité pokrytie definičného intervalu – pre každé $i>1$ sa za dolnú hranicu intervalu INT_i dosadí horná hranica intervalu INT_{i-1}

Ďalšie transformácie

V tejto fáze sa predpokladá, že dátam rozumieme, máme ich vyčistené a sú viac-menej pripravené na modelovanie. Najskôr sa hľadajú ***odvodené (derivované) atribúty***. Odvodený atribút je vlastne nový atribút, ktorý sa vytvára kombináciou atribútov za pomoci agregácií alebo delením týchto

atribútov. Obidve metódy vyžadujú vysokú úroveň porozumenia dátam a značnú schopnosť pochopiť potreby zákazníka.

Agregované hodnoty sa vytvárajú sčítaním, odčítaním alebo spriemerovaním podľa určitého atribútu, napríklad vytváranie súčtov tržieb za dni alebo priemerných mesačných tržieb podľa rokov. *Delenie* je proces, pri ktorom komplexný atribút transformujeme na jeden alebo viac atribútov, ktorých hodnoty obsahujú parciálne informácie pôvodného atribútu. Ako príklad uvediem vytvorenie 3 atribútov – deň, mesiac a rok narodenia – z atribútu obsahujúceho rodné čísla.

3.4. Modelovanie

Modelovanie pozostáva z aplikácie niektorého z algoritmov pre dobývanie znalostí na pripravené dáta. Snahou v tomto kroku je vytvoriť model, ktorý by naplnil stanovené ciele a dal odpovede na otázky. Dôležitá je voľba správneho algoritmu, ktorá závisí od povahy dát, úlohy a cieľov. Častý je prípad, kedy sa použije viacero metód a na záver sa výsledky porovnávajú, prípadne kombinujú. Jednotlivé modelovacie techniky a algoritmy sú podrobne rozobrané v kapitole 4.

3.5. Vyhodnotenie výsledkov

V tejto fáze sa dosiahnuté výsledky interpretujú – upravujú sa do vhodnej podoby, ktorá je pochopiteľná pre manažérov zadávateľa úlohy. Overí sa, či dosiahnuté výsledky naplnili očakávania a ciele, ktoré boli stanovené. Hlavným zdrojom informácií pre túto kapitolu bola publikácia [5].

Testovanie

Dôležitou súčasťou tohto kroku je ***testovanie modelu***. V prípade klasifikácie používajúcej metódu učenia s učiteľom máme k dispozícii správne klasifikované dáta. Môžeme teda použitím niektorého z kritérií otestovať správnosť modelu a porovnávať výsledky pre rôzne modelovacie algoritmy a odlišné parametre na rovnakých dátach. Potrebujeme dáta, ktoré sa použijú na tvorbu modelu (trénovacie) a dáta, ktoré za použitia vytvoreného modelu ohodnotíme a porovnáme s očakávanou klasifikáciou (testovacie). Voľba týchto dát sa dá vykonať rôznymi spôsobmi:

a) ***Zhodné dáta*** – na trénovanie a následné testovanie sa použijú rovnaké dáta. Pri modelovaní však často dochádza k preučeniu[6], ktoré ale môže byť v tomto prípade kvôli dobrým výsledkom testov prehliadnuté. Preto sa tento spôsob voľby trénovacej a testovacej množiny skoro nepoužíva.

[6] Preučenie je stav, pri ktorom nadobudnuté znalosti vystihujú skôr náhodné charakteristiky trénovacích dát a neodhalia to podstatné, čo je možné použiť na generalizáciu. ([5], str.224, odst.2).

b) ***Krížová validácia*** – tiež nazvaná k-násobná krížová validácia. Dáta sa dopredu rozdelia na *k* približne rovnakých častí. Potom nasleduje *k* behov modelovania a testovania. Na testovanie sa volí vždy iná skupina a zvyšných *k-1* sa použije na trénovanie. Výsledky behov sa spriemerujú. Je vhodné, aby každá časť obsahovala zhruba rovnaký pomer zástupcov jednotlivých tried. Preto je dobré vybrať do každej skupiny pre každú triedu náhodne toľko záznamov z tejto triedy, aby bol pomer zachovaný.

c) ***Metóda „leave-one-out“*** – je špeciálnym prípadom krížovej validácie. Ako *k* sa volí počet všetkých záznamov. Pri veľkom množstve dát môže byť táto metóda náročná na čas.

d) ***Metóda „bootstrap“*** – špecifikom tejto metódy je, že sa príklady použité na učenie môžu opakovať. Keď máme k dispozícii *n* príkladov, tak na vytvorenie trénovacej množiny vykonáme n-krát výber s vrátením.

e) ***Náhodný výber*** – náhodne sa vyberie zvolené percento dát na trénovanie a zvyšok sa použije na testovanie. Trénovanie aj testovanie sa vykoná iba raz a každý príklad je použitý práve raz. Časté je rozdelenie 75:25 alebo 80:20.

Výsledok testovania sa zaznamená do *matice zámen*. Nech *C* je kategoriálny cieľový atribút, ktorý nadobúda hodnôt $c_1,...,c_k$. Potom matica zámen je matica Z_{kxk}, kde celočíselný prvok $z_{i,j}$ obsahuje počet príkladov, ktoré patria do triedy c_i a model ich zaradil do triedy c_j. Počet správne klasifikovaných testovacích záznamov pre jednotlivé triedy sa teda nachádza na diagonále.

$$Z(M_1) = \begin{pmatrix} 127 & 3 \\ 42 & 98 \end{pmatrix} \qquad Z(M_2) = \begin{pmatrix} 118 & 12 \\ 10 & 130 \end{pmatrix}$$

obr. 3 – Matice zámen pre modely M_1 a M_2

Niekedy je vhodné vykonať vážené testovanie. Chybné rozhodnutia sú v praxi málokedy ekvivalentné. Predstavme si model, ktorý má na základe charakteristík pacienta (vek, váha, ...) a výsledkov testov (tlak, pulz, krvný cholesterol, ...) rozhodnúť, či je pacient kandidátom na infarkt. Ak model nesprávne označí zdravého človeka, ten sa zbytočne podrobí ďalším vyšetreniam, ktoré ho stoja čas a zdravotnú poisťovňu peniaze. Opačná chyba ale môže mať za následok ľudský život. Preto je vhodné pri porovnávaní modelov navrhnúť testovanie tak, aby každá chyba mala určenú cenu, ktorú musíme pri tomto omyle „zaplatiť“. Tieto ceny sa zaznamenajú do *matice cien* R_{kxk}, ktorá je podobná matici zámen. Prvok matice $r_{i,j}$ je reálne číslo, ktoré určuje cenu za chybnú klasifikáciu záznamu z triedy c_i do triedy c_j. Kým matica zámen je špecifická pre každý model, matica cien sa viaže na dáta respektíve problematiku.

$$R = \begin{pmatrix} 0 & 1.8 \\ 0.4 & 0 \end{pmatrix}$$

obr. 4 – Matica cien (spoločná pre všetky modely pre dané dáta)

Charakteristiky hodnotenia

Nech máme dáta a maticu cien, ktorá bola zostavená expertom v danej problematike. Použitím rozdielnych modelovacích metód sme vytvorili na rovnakých dátach 2 modely – M_1 a M_2. Pomocou týchto modelov sme klasifikovali testovacie dáta a dostali sme výsledky, ktoré môžeme vidieť v maticiach zámen na obr. 3. Na ohodnotenie, ktorý z modelov je lepší, môžeme použiť niekoľko rôznych charakteristík.

Prvou z nich je ***celková správnosť*** *Acc*. Celková správnosť je pomer správnych zaradení do triedy ku veľkosti testovacej množiny. Pri použití matice zámen je to súčet prvkov na diagonále ku súčtu všetkých prvkov matice. Podobnou charakteristikou je ***celková chyba*** *Err*. Tú vypočítame ako pomer nesprávne klasifikovaných záznamov ku veľkosti testovacej množiny. Za predpokladu, že neuvažujeme špeciálne nakladanie s kontradikciami[7], platí *Err = 1 – Acc*. V prípade našich modelov vychádza táto charakteristika v prospech druhého modelu:

$$Acc(M_1) = \frac{225}{270} = \underline{\underline{83.33\,\%}} \qquad Acc(M_2) = \frac{248}{270} = \underline{\underline{91.85\,\%}}$$

Ak berieme do úvahy aj maticu cien, môžeme spočítať pre model ***váženú chybu***. Tú spočítame nasledovne:

$$Err(R, M) = \sum_{i=1}^{k} \sum_{\substack{j=1 \\ i \neq j}}^{k} r_{i,j} m_{i,j}$$

Vidíme, že vážená chyba vychádza lepšie pre model M_1. Aj keď celková chyba je u tohto modelu väčšia, prípadná strata (cena, ktorú musíme zaplatiť za zlé rozhodnutia - najčastejšie nás zaujíma finančná) vychádza u tohto modelu lepšie.

$$Err(R, M_1) = \underline{\underline{22.2}} \qquad Err(R, M_2) = \underline{\underline{25.6}}$$

Ak je zastúpenie tried v dátach silne nerovnomerné, je vhodné použiť ***priemernú správnosť*** alebo ***priemernú chybu***. Pre každú triedu c_i sa

[7] Kontradikcie sú protirečivé záznamy. Ak existuje v dátach skupina záznamov, ktoré majú zhodné vstupné atribúty a líšia sa v cieľovom atribúte, nemôže model nikdy dosiahnuť 100% správnosť resp. 0% chybu. Preto sa niekedy tieto záznamy vynechávajú zo štatistík.

spočíta celková správnosť triedy *Acc(c_i,M)*, ktorá určuje počet správnych klasifikácii do tejto triedy ($z_{i,i}$) ku celkovému počtu záznamov v tejto triede. Výsledok sa vypočíta ako priemer z týchto hodnôt. Opäť môžeme priemernú chybu určiť ako doplnok priemernej správnosti:

$$\Delta Err(M) = 1 - \Delta Acc(M) = 1 - \frac{\sum_{i=1}^{k} Acc(c_i, M)}{k}$$

$$Acc(c_i, M) = \frac{z_{i,i}}{\sum_{j=1}^{k} z_{i,j}}$$

Krivka učenia

Krivka učenia (*learning curve*) vyjadruje, ako sa vyvíja kvalita (napr. celková spoľahlivosť) modelu v závislosti na veľkosti trénovacej množiny alebo na dĺžke učenia. Predpokladom je, že s rastúcim počtom záznamov v trénovacej množine (resp. dĺžkou učenia) bude rásť aj kvalita modelu. Z tohto grafu však často môžeme odhaliť, že pre daný typ modelu je už od istého bodu nárast minimálny, dokonca môže začať kolísať. Z krivky učenia tiež môžeme identifikovať preučenie. Ako vyzerá krivka učenia môžeme vidieť na obr. 5.

obr. 5 – Príklad krivky učenia

V prípade neurónových sietí môže krivka učenia slúžiť k tomu, aby sme rozhodli, kedy je možné proces učenia ukončiť.

Vizualizácia

Rôzne formy vizualizácie sa využívajú nielen vo fáze porozumenia dát, ale aj na zobrazenie a vyhodnotenie výsledkov, či samotného modelu. ***Vizualizácia modelu*** je pri niektorých typoch modelu nenahraditeľnou pomôckou a často umožní lepšie pochopiť nadobudnuté znalosti. Typickým príkladom sú rozhodovacie stromy. Tie možno reprezentovať vo forme klasifikačných pravidiel, alebo použiť jednoduchý obrázok zobrazujúci jeho stromovú štruktúru. Pridaním farieb môžeme napríklad odlíšiť jednotlivé listy podľa tried.

V prípade analýzy nákupného košíka tiež môžeme výslednú tabuľku pravidiel sprehľadniť pridaním farieb, ktoré vyjadrujú mieru podpory a spoľahlivosti pravidla. Zaujímavú formu reprezentácie asociačných pravidiel uvádza publikácia [5], kedy sa model zobrazí ako úplný graf a hrúbka/farba hrany určuje, aký silný je vzťah medzi danými hodnotami.

3.6. Aplikácia výsledkov

Keď už máme hotový model, ktorého kvalita a použiteľnosť bola overená, zostáva nám nadobudnuté znalosti aplikovať, aby plnili prvotný zámer. Konkrétna aplikácia výsledkov je závislá na požiadavkách zadávateľa a type získaného modelu. Úlohy pre naplnenie tohto kroku vyžadujú kooperáciu alebo sú plne v moci zadávateľa.

V prípade neurónovej siete je potrebné vytvoriť rozhranie, ktoré umožní použiť vytvorenú „rozhodovaciu čiernu skrinku“ na získavanie odpovedí. Ak je ako modelovacia technika použitá analýza nákupného košíka, zadávateľ získa pravidlá, ktoré mu pomôžu lepšie sa rozhodovať pri vytváraní akčných ponúk a reorganizácii obchodu. Je teda nutné stanoviť, v akej forme bude získaný model uložený a použiteľný.

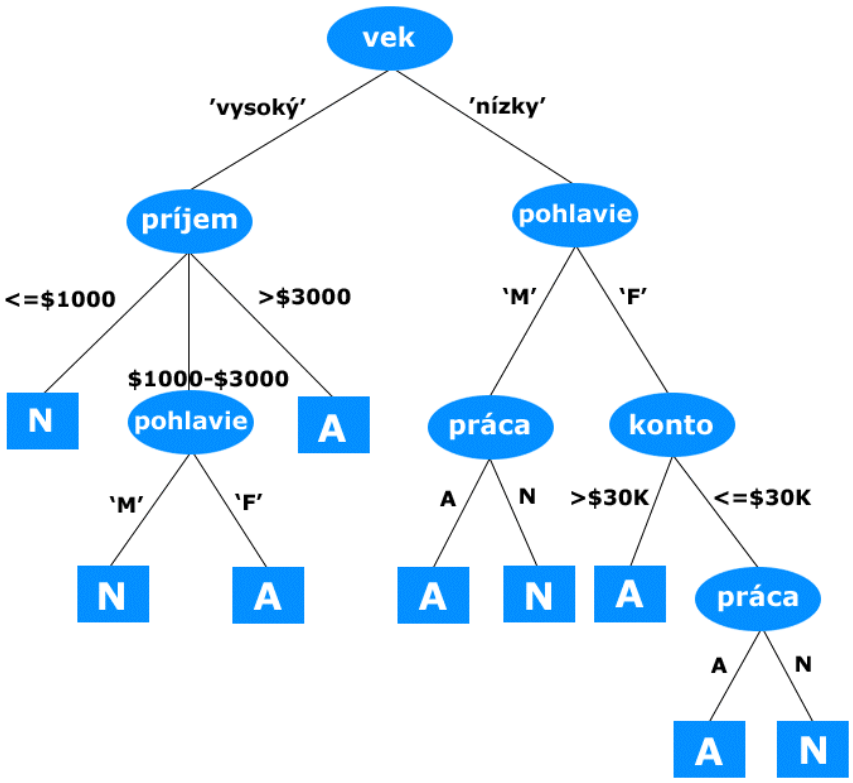
4. Modelovacie techniky a algoritmy

Táto kapitola obsahuje popis metód a algoritmov používaných pri vytváraní analytických modelov. Pod pojmom ***model*** budeme chápať dátovú štruktúru, ktorá uchováva nadobudnuté znalosti a umožňuje ich použitie pre splnenie stanovených cieľov. Zameriam sa na algoritmy, ktoré umožňujú konštrukciu modelov použiteľných na klasifikáciu a získavanie asociačných pravidiel. Medzi najznámejšie patria rozhodovacie stromy, neurónové siete, Bayesovský klasifikátor, metóda k-najbližších susedov a metóda analýzy nákupného košíka.

4.1. Rozhodovacie stromy

Rozhodovací strom je analytický model použiteľný na klasifikáciu, prípadne predikciu. Má podobu matematického stromu. Každý vnútorný uzol stromu reprezentuje jeden z atribútov. Na hranách medzi týmto uzlom a jeho potomkami sa nachádzajú v prípade spojitého atribútu intervaly, v prípade kategoriálneho atribútu jednotlivé kategórie. Pre každý vnútorný uzol platí, že kategórie resp. intervaly na týchto hranách pokrývajú všetky možné hodnoty atribútu a sú odlišné resp. disjunktné v prípade intervalov. Strom je zakorenený a v listoch sa nachádzajú hodnoty z domény cieľového atribútu.

Vnútorné uzly sa volajú rozhodovacie, pretože sa podľa nich rozhoduje, do ktorého podstromu pokračovať na základe hodnoty daného atribútu pre konkrétny záznam. Použitie vytvoreného a naučeného rozhodovacieho stromu na klasifikáciu je nasledovné: Začneme v koreni, na základe hodnôt atribútov postupujeme v strome, kým sa nedostaneme do listu, ktorý nám poskytne klasifikáciu predloženého vstupného záznamu.



obr. 6 – Ukážka rozhodovacieho stromu

Na obrázku vidíme príklad rozhodovacieho stromu. Najselektívnejším atribútom v tomto prípade je „vek“, cieľovým atribútom je „hypotéka“, pričom vstupný záznam môžeme klasifikovať do 2 tried. Atribúty „príjem“ a „konto“ sú spojité, ostatné atribúty sú kategoriálne.

Klasifikačné stromy sú rozhodovacie stromy, ktoré majú kategoriálny cieľový atribút, takže záznamy sú zaraďované do tried. Rozhodovacie stromy, ktoré predpovedajú hodnotu spojitého cieľového atribútu sa nazývajú ***regresné stromy***.

Silnou stránkou rozhodovacích stromov je fakt, že sú reprezentované pravidlami, ktoré je možné vyjadriť prirodzeným jazykom, takže sú ľahko pochopiteľné a použiteľné. Toto je veľký rozdiel oproti neurónovým sieťam, kde model má pre nás podobu čiernej skrinky.

Na vytvorenie rozhodovacieho stromu existuje rada algoritmov. Medzi najznámejšie patria CART, CLS, ID3, GID3, C4.5 a CHAID.

Algoritmy induktívneho učenia.

Táto skupina algoritmov sa používa na vytváranie rozhodovacích stromov systémom zhora nadol. Používa princíp „Rozdeľuj a panuj“. Proces vytvárania stromu je induktívny. Znamená to, že sa snaží z existujúcich záznamov odvodiť vlastnosti budúcich záznamov. Do tejto skupiny patria

napríklad algoritmy ID3, ID5R alebo C4.5. Pre túto skupinu algoritmov sa zaužívala skratka TDIDT[8].

Základ obecného algoritmu v pseudokóde vyzerá nasledovne:

1. Vytvor koreňový uzol reprezentujúci všetky tréningové záznamy.
2. Ak patria všetky záznamy v zvolenom uzle do rovnakej triedy (majú rovnakú hodnotu cieľového atribútu), označ uzol ako list s hodnotou cieľového atribútu týchto záznamov.
3. Inak vyber vhodný atribút *A*, podľa ktorého bude uzol rozdeľovať záznamy.
4. Pre každú hodnotu a_i z domény zvoleného atribútu *A*[9] vytvor nového potomka a priraď mu tie záznamy z deleného uzla, pre ktoré atribút A má hodnotu a_i.
5. Ak nie je splnená ukončovacia podmienka, vyber jeden z nelistových uzlov a prejdi na krok 2.

Medzi špecifiká jednotlivých algoritmov v tejto skupine patrí spôsob voľby deliaceho atribútu, ukončovacia podmienka a postup pri sporných alebo nejednoznačných záznamoch. Spornými záznamami rozumieme skupinu aspoň 2 záznamov, ktoré majú identické hodnoty vstupných atribútov a odlišnú hodnotu cieľového atribútu. V prípade nejednoznačnosti záznamov sa často uplatňuje systém väčšinovej voľby.

Shannonova teória informácie

Claude Shannon vyslovil v roku 1949 teóriu, ktorá sa dodnes používa nielen v oblasti komunikačných technológií, ale aj pri vytváraní rozhodovacích stromov. Táto teória hovorí, že keď máme správy $\boldsymbol{x_1,\ldots,x_n}$ a pravdepodobnostné rozdelenie $\boldsymbol{p}$, ktoré každej správe priraďuje pravdepodobnosť výskytu a pre ktoré platí:

$$\sum_{i=1}^{n} p(x_i) = 1,$$

potom entrópiou H, ktorá odpovedá *neusporiadanosti* daného súboru správ rozumieme:

$$H = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i).$$

[8] Skratka TDIDT nahradzuje anglické Top-Down Induction of Decision Trees, označuje skupinu algoritmov na generovanie rozhodovacích stromov, ktoré budujú strom induktívne a zhora nadol.

[9] Algoritmus C4.5 pripúšťa spojitý deliaci atribút. V takom prípade sa záznamy rozdelia podľa najlepšieho deliaceho bodu na 2 skupiny.

Hore uvedený vzťah sa nazýva ***Shannonova entrópia***. Ako sa píše v [2]: „Čím je entrópia správ vyššia, tým menej určitý je obsah budúcej správy a tým väčšie množstvo informácie získame, keď správu prijmeme.“

Ukážeme si, ako sa počíta entrópia a z nej odvodený informačný zisk pre jednotlivé uzly a atribúty. Nech máme uzol *U*, ktorému patrí n trénovacích záznamov. Nech *A* je atribút, ktorý môže nadobúdať hodnoty $a_1,...,a_j$ a nech máme $m_1,...,m_j$, kde m_i vyjadruje počet záznamov, ktoré majú atribút *A* s hodnotou a_i. Nech *C* je cieľový atribút, ktorý môže nadobúdať hodnôt $c_1,...,c_k$ a nech máme $n_1,...,n_k$ a $n_1(A=a_i),...,n_k(A=a_i)$ pre každé *i* od 1 do *j*, kde n_l označuje počet trénovacích záznamov patriacich do triedy c_l a $n_l(A=a_i)$ je počet trénovacích záznamov náležiacich do triedy c_l, ktoré majú atribút *A* s hodnotou a_i. Potom označíme strednú entrópiu atribútu *A* v uzle *U* ako:

$$H(U,A)=\sum_{i=1}^{j}\frac{m_i}{n}H\left(A=a_i\right).$$

Výraz $H(A=a_i)$ označuje entrópiu pre atribút *A* a jeho hodnotu a_i, ktorú spočítame nasledovne:

$$H\left(A=a_i\right)=-\sum_{l=1}^{k}\frac{n_l\left(A=a_i\right)}{m_i}\log_2\frac{n_l\left(A=a_i\right)}{m_i}$$

Informačný zisk pre uzol *U* a atribút *A* spočítame nasledovne:

$$I\left(U,A\right)=H\left(U\right)-H\left(U,A\right)$$

Výraz $H(U)$ v tomto prípade označuje entrópiu súboru trénovacích záznamov prislúchajúcich uzlu *U*, ktorú spočítame ako:

$$H\left(U\right)=-\sum_{i=1}^{k}\frac{n_i}{n}\log_2\frac{n_i}{n}$$

Zo vzorcov je zrejmé, že uzol, pre ktorý sú všetky záznamy z rovnakej triedy, bude mať entrópiu nulovú. Na druhej strane, ak obsahuje záznamy z viacerých tried, bude mať nenulovú entrópiu pre každý deliaci atribút *A*. Aby sa naplnil cieľ vytvoriť minimálny rozhodovací strom, vyberá sa pre každý uzol atribút s najmenšou entrópiou – teda s maximálnym možným informačným ziskom.

Algoritmus ID3

Algoritmus ID3 (*Iterative Dichotomizer 3*) je rozšírením základného algoritmu CLS (*Concept Learning System*), ktorý vyvinul v roku 1979 Ross Quinlan na univerzite v Sydney. Na výber najlepšieho deliaceho atribútu využíva Shannonovu teóriu informácie, ktorá meria informačný zisk pomocou

entrópie. Algoritmus pracuje len s kategoriálnymi atribútmi, preto je nutné spojité atribúty kategorizovať.

Uzol sa stane listovým aj vtedy, keď:

a) nespadajú do neho žiadne záznamy – vtedy sa hodnota cieľového atribútu vyberá väčšinovo zo všetkých záznamov.

b) na ceste z koreňa do tohto uzlu boli použité na selekciu všetky atribúty. Množina dát je v tomto prípade nedostatočná alebo protirečivá. Buď sa zvolí klasifikácia väčšinovým princípom zo záznamov patriacich tomuto uzlu, alebo algoritmus končí.

Algoritmus končí, ak neexistujú žiadne nelistové uzly.

Algoritmus ID5R

Algoritmus ID5R *(Inductive Dichotomizer 5 Recursive*) je iteratívnou modifikáciou algoritmu ID3. Jeho hlavnou výhodou je schopnosť opätovne upravovať strom pri príchode nových dát. V prípade použitia ID3 sa vždy pri príchode nových dát musí znovu generovať celý rozhodovací strom a strojový čas potrebný na túto operáciu narastá úmerne s veľkosťou dát.

Podobne ako ID3, aj tento algoritmus používa na voľbu najvhodnejšej deliacej premennej v uzle informačný zisk. Autor v [4] píše, že pre rovnakú množinu trénovacích príkladov dostaneme strom identický ako pri použití ID3.

Algoritmus ID5R predpokladá uchovávanie pomocných dát pre každý uzol. Tieto pomocné dáta sú pri tvorbe vhodne inicializované a analogicky upravované v jednotlivých iteráciách. Pomocné informácie umožňujú algoritmu overiť, či v uzle daný atribút poskytuje maximálny informačný zisk a nájsť vhodnejší atribút, ak tomu tak nie je. Pri zmene deliaceho atribútu v uzle je potrebná rekonštrukcia podstromu daného týmto uzlom.

Preto sa tento algoritmus hodí hlavne v prípadoch, kedy je potrebné na základe aktuálnej množiny trénovacích dát vygenerovať použiteľný rozhodovací strom a na základe odozvy v podobe nových dát tento strom vylepšovať. Takúto úlohu nazývame úlohou typu *on-line*. V opačnom prípade autor odporúča použiť algoritmus ID3.

Algoritmus C4.5

Algoritmus C4.5 (Ross Quinlan) je tiež odvodený z algoritmu ID3. Na rozdiel od jeho rekurzívneho potomka zostáva neinkrementálnym algoritmom. Jeho hlavnou výhodou je možnosť použiť na vetvenie spojitý vstupný atribút. Na výber najvhodnejšieho vetviaceho atribútu používa ***pomerové kritérium zisku***. Podľa [2] ho táto voľba zvýhodňuje oproti algoritmom používajúcim klasický informačný zisk, ktorý uprednostňuje výber testovacieho atribútu s väčšou doménou.

Samotný algoritmus C4.5 je tak ako aj ID3 odvodený od základného algoritmu pre rozhodovacie stromy budované zhora nadol. V prípade spojitého atribútu pozostáva testovacia podmienka z prahovej hodnoty, ktorá rozdelí trénovaciu množinu pre daný uzol na 2 skupiny. Voľba prahovej hodnoty sa vykonáva tak, že sa vytvorí postupnosť prítomných hodnôt daného atribútu

{a₁, a₂, …, aₙ}. Hľadá sa taký index ***i***, ktorý najlepšie rozdelí túto postupnosť na 2. Zvolenou ohodnocujúcou funkciou sa otestuje všetkých *n-1* rozdelení a vyberie sa to najlepšie. Ako kritérium sa používa informačný zisk alebo pomerový informačný zisk.

Pomerové kritérium zisku

Nech máme uzol *U* a atribút *A*, ktorý nadobúda ***n*** hodnôt: ***a₁,…,aₙ***. Nech ***p*** je pravdepodobnostné rozdelenie, ktoré priradí každej hodnote pravdepodobnosť jej výskytu. Pomerovou entrópiou označíme:

$$H_p(U,A) = -\sum_{i=1}^{n} p(a_i)\log_2 p(a_i)$$

Nech ***I(U,A)*** je informačný zisk pre uzol *U* a atribút *A*, ktorý bol popísaný na strane 27. Potom pomerovým informačným ziskom pre uzol *U* a atribút *A* označíme:

$$I_p(U,A) = \frac{I(U,A)}{H_p(U,A)}$$

Publikácia [2] tvrdí, že použitie pomerového informačného zisku vedie ku generovaniu rozmerovo menších rozhodovacích stromov.

Algoritmus CART

Algoritmus CART (*Classification and Regression Tree*) je algoritmus, ktorý bol pôvodne navrhnutý v roku 1984 skupinou ľudí okolo Breimana: Breiman, Friedman, Olshen a Stone. Z názvu je jasné, že podporuje klasifikačné aj regresné stromy. Znamená to, že vstupné atribúty aj cieľový atribút môžu byť kategoriálne alebo spojité. Strom vytvorený pomocou algoritmu CART je binárny rozhodovací strom a na ceste z koreňa do listu sa môže jeden atribút nachádzať vo viacerých deliacich uzloch.

Základná idea algoritmu je analogická s obecným algoritmom pre TDIDT. Cieľom je v každom uzle vybrať delenie, ktoré maximalizuje zvolené deliace kritérium. Algoritmus rozlišuje 2 typy kategoriálnych atribútov: *nominálne kategoriálne* a *ordinálne kategoriálne*. Ordinálny kategoriálny atribút je kategoriálny atribút, u ktorého ma zmysel uvažovať poradie tried. Na základe toho stačí nájsť pri hľadaní delenia deliaci bod a nemusia sa vyhodnocovať všetky podmnožiny.

V každom uzle sa otestujú možné delenia podľa všetkých vstupných atribútov. U nominálneho kategoriálneho atribútu s *N* triedami existuje 2^{N-1} možných delení, u ordinálneho kategoriálneho a spojitého atribútu s *M* odlišnými hodnotami v dátach je to *M-1* delení. Algoritmus vyzerá nasledovne:

1. Pre každý vstupný atribút nájdi najlepšie delenie. Ak je atribút spojitý alebo ordinálny kategoriálny, utrieď vzostupne jeho hodnoty. Postupne preskúmaj všetky možné deliace body

a získaj ten s maximálnou hodnotou deliaceho kritéria. Ak je atribút nominálny kategoriálny, preskúmaj všetky delenia tried na 2 navzájom disjunktné podmnožiny a vyber to s maximálnu hodnotu deliaceho kritéria.

2. Nájdi najlepšie delenie pre uzol – z najlepších delení pre atribúty sa vyberie najlepšie celkové delenie.
3. Ak bola splnená ukončovacia podmienka, skonči.
4. Rozdeľ uzol podľa najlepšieho delenia a dáta priraď do nových uzlov.

Ak je cieľový atribút kategoriálny, používa sa ako deliace kritérium niektoré z nasledujúcich kritérií: ***Giniho kritérium***, ***Twoing Criterion*** alebo ***Ordered Twoing Criterion***. V prípade, že je spojitý, používa sa tzv. ***Least Squares Deviation (LSD)***.

Ukončovacia podmienka je jedna z nasledujúcich:

a) Ak uzol obsahuje len identické hodnoty cieľového atribútu.
b) Ak všetky záznamy v uzle majú identické hodnoty vstupných atribútov.
c) Ak počet záznamov patriacich uzlu je menší ako minimum definované užívateľom.
d) Ak hodnota deliaceho kritéria pre najlepšie delenie je menšia ako minimum definované užívateľom.
e) Ak bola dosiahnutá maximálna hĺbka stromu definovaná užívateľom.

Deliace kritéria

Giniho kritérium je založené na Gini indexe, ktorý zostrojil taliansky štatistik Conrad Gini v roku 1912. Nech máme vstupné atribúty $A_1,...,A_m$, cieľový atribút C s hodnotami $c_1,...,c_J$, celú trénovaciu množinu $\hbar = \{\vec{x}_n, y_n\}_{n=1}^{N}$, trénovaciu množinu pre uzol t $\hbar(t)$, váhu w_n a frekvenciu f_n asociovanú s n-tým záznamom trénovacej množiny. Pre $j=1,...,J$ označíme:

- $\pi(j)$ – pravdepodobnosť, že C je rovné c_j
- $p(j,t)$ – pravdepodobnosť, že záznam je v uzle t a triede c_j
- $p(t)$ – pravdepodobnosť, že záznam je v uzle t
- $p(j|t)$ – pravdepodobnosť, že záznam je v triede c_j, za predpokladu, že spadol do uzla t

Uvedené pravdepodobnosti spočítame ako:

$$p(j,t) = \frac{\pi(j) N_{w,j}(t)}{N_{w,j}}$$

$$p(t) = \sum_j p(j,t)$$

$$p(j \mid t) = \frac{p(j,t)}{p(t)} = \frac{p(j,t)}{\sum_{j'} p(j',t)},$$

kde

$$N_{w,j} = \sum_{n \in \hbar} w_n f_n I(y_n = c_j)$$

$$N_{w,t}(t) = \sum_{n \in \hbar(t)} w_n f_n I(y_n = c_j)$$

pričom *I(a=b)* je funkcia, ktorá nadobúda hodnotu 1 pre *a* = *b*, inak 0.

Potom ***Giniho miera znečistenia*** *g(t)* v uzle *t* a ***Giniho deliace kritérium*** (zníženie znečistenia) *Δg(s,t)* sú definované ako:

$$g(t) = \sum_{i,j} p(i \mid t) p(j \mid t)$$

$$\Delta g(s,t) = g(t) - p_L g(t_L) - p_R g(t_R),$$

kde p_L/p_R je pravdepodobnosť, že záznam pôjde do ľavého/pravého potomka t_L/t_R uzlu *t*. Tieto pravdepodobnosti spočítame nasledovne:

$$p_L = \frac{p(t_L)}{p(t)} \qquad p_R = \frac{p(t_R)}{p(t)}.$$

LSD kritérium

Ako už bolo spomenuté, toto kritérium sa používa ako deliace v prípade spojitého cieľového atribútu. Nech máme trénovaciu množinu $\hbar = \{\vec{x}_n, y_n\}_{n=1}^N$, trénovaciu množinu pre uzol t $\hbar(t)$, váhu w_n a frekvenciu f_n asociovanú s n-tým záznamom trénovacej množiny. ***LSD kritérium*** *Δi(s,t)* s mierou LSD znečistenia *i(t)* definujeme ako:

$$\Delta i(s,t) = i(t) - p_L i(t_L) - p_R i(t_R)$$

$$i(t) = \frac{\sum_{n \in \hbar(t)} w_n f_n \left(y_n - \bar{y}(t)\right)^2}{\sum_{n \in \hbar(t)} w_n f_n},$$

kde $\bar{y}(t)$ je priemerná hodnota spojitého cieľového atribútu v uzle *t* a p_L/p_R je pravdepodobnosť, že záznam pôjde do ľavého/pravého potomka t_L/t_R uzlu *t*. Tieto hodnoty spočítame nasledovne:

$$p_L = \frac{N_w(t_L)}{N_w(t)}, \quad p_R = \frac{N_w(t_R)}{N_w(t)}, \quad \bar{y}(t) = \frac{\sum_{n \in \hbar(t)} w_n f_n y_n}{N_w(t)},$$

pričom

$$N_w(t) = \sum_{n \in \hbar(t)} w_n f_n .$$

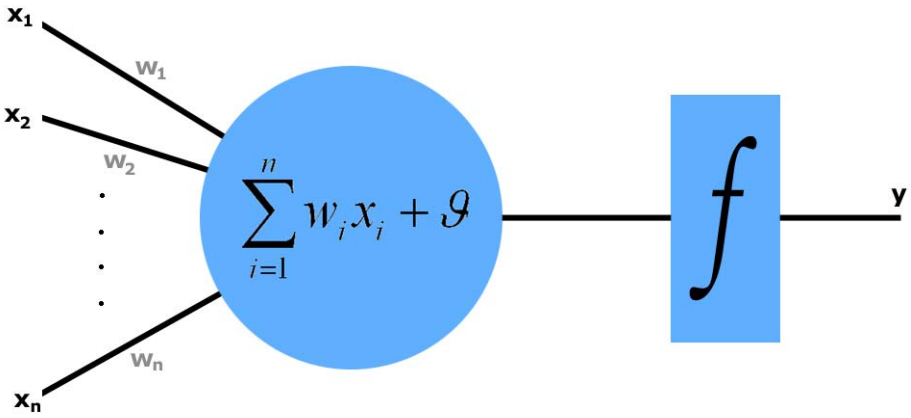
4.2. Neurónové siete

Neurónové siete sú jednou z najpoužívanejších metód v oblasti umelej inteligencie. Označované sú tiež ako ***umelé neurónové siete***. Ich fungovanie vytvára určitú analógiu s biologickými neurónovými sieťami – mozgom. Základnými stavebnými jednotkami neurónových sietí sú neuróny. Jednotlivé neuróny sú pospájané orientovanými spojnicami - synapsiami.

Ako tvrdia autori knihy [3], neurónové siete sú nevýhodné kvôli slabej interpretovateľnosti nadobudnutého modelu. Na druhej strane vidia ich veľkú výhodu v tom, že majú vysokú toleranciu „zašumených“ dát a sú schopné klasifikovať vzory, na ktorých neboli vytrénované.

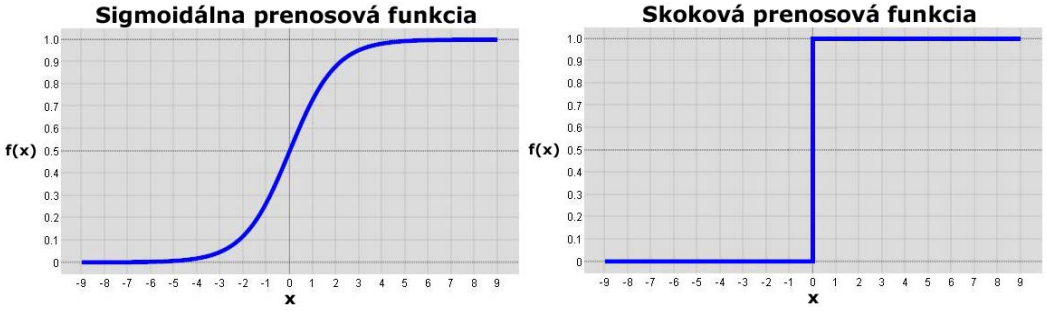
Každý neurón má **n** vstupov, ktoré sú spravidla reprezentované ako reálny vektor $\vec{x} \in R^n$. Neurón má práve jeden reálny výstup *y*, pričom tento výstup môže byť rozvedený na vstup viacerých neurónov. Jednotlivé synapsie sú ohodnotené váhami $w_1,...,w_n$, ktoré sú typicky reprezentované ako reálne čísla. Model neurónu môžeme vidieť na obr. 7.

Dôležitá vlastnosť umelých neurónových sietí je schopnosť učiť sa. Učenie neurónovej siete delíme na *učenie s učiteľom* a *učenie bez učiteľa*. V prvom prípade prebieha učenie formou adaptácie synaptických váh v závislosti od skutočného a požadovaného výstupu. Pri učení bez učiteľa je požadovaný výstup neznámy a sieť vstupné vzory triedi do skupín a reaguje na typického zástupcu.



obr. 7 – Model neurónu

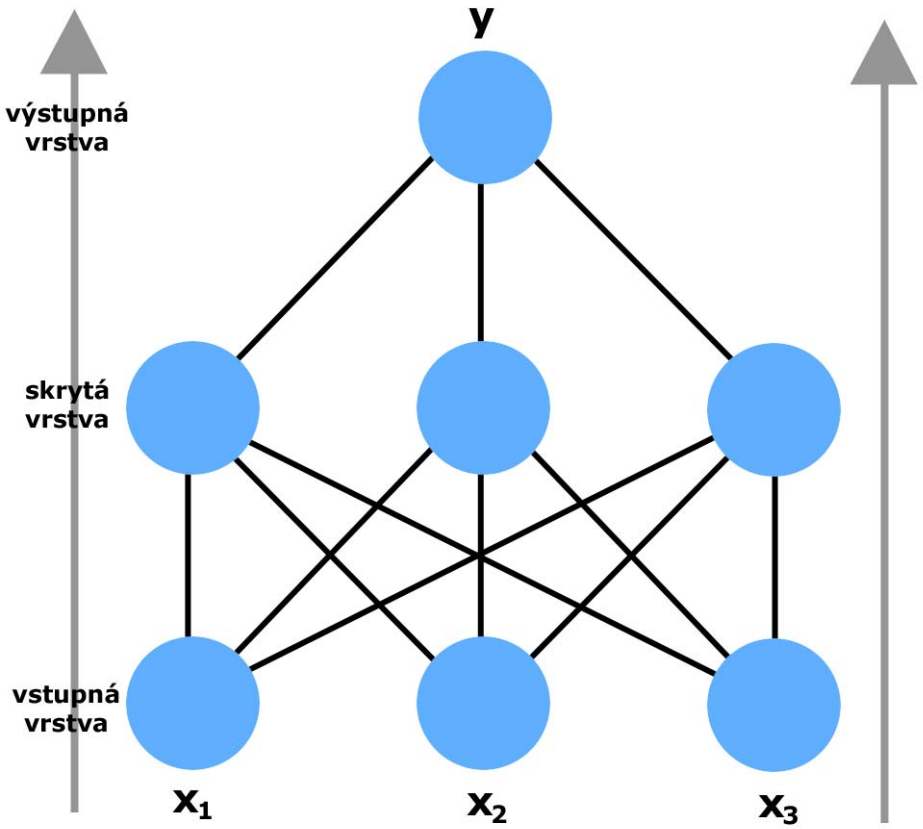
Najjednoduchšia neurónová sieť sa nazýva perceptronová. Je tvorená jedným neurónom - ***perceptronom***. Táto sieť počíta svoj výstup ako vážený súčet vstupov, ktorý sčíta s prahovou hodnotou a výsledok predáva ďalej pomocou nelineárnej prenosovej funkcie. Najčastejšie sa používa skoková prenosová funkcia alebo sigmoidálna prenosová funkcia. Z obrázku týchto funkcií je jasné, že použitie skokovej prenosovej funkcie vedie k binárnym výstupom a sigmoidálnej k spojitým výstupom.



obr. 8 – Prenosové funkcie

Vrstevnaté neurónové siete sú pomerne veľkou skupinou umelých neurónových sietí. Neuróny v tejto sieti sú usporiadané do ***k*** vrstiev, kde ***k≥2***. Prvá vrstva je označená ako vstupná a k-tá ako výstupná. Vnútorné vrstvy sa nazývajú skryté. Počet vrstiev siete sa počíta ako počet vrstiev s výstupnými neurónmi, teda pre sieť s jednou skrytou vrstvou je celkový počet vrstiev siete 2. Počet skrytých vrstiev môže byť ľubovoľný, ale podľa skúseností autorov [3] je najčastejšie práve jedna.

Vrstevnaté neurónové siete sú reprezentované pomocou orientovaných acyklických grafov, kde neuróny tvoria vrcholy tohto grafu, hrany odpovedajú synapsiám a ich ohodnotenie synaptickým váham. Každé 2 po sebe nasledujúce vrstvy tvoria úplný bipartitný graf, t.j. výstup každého neurónu ***i*** je spojený so vstupom každého neurónu ***j*** na nasledujúcej vrstve, okrem výstupnej vrstvy. Váhu synapsie medzi neurónmi i a j označíme $w_{i,j}$.



obr. 9 – Vrstevnatá neurónová sieť

Formálne sa dá neurónová sieť vyjadriť ako usporiadaná 6-tica ***M=(N,C,I,O,w,t)***, kde:

- *N* je konečná množina neurónov, ktoré sú usporiadané do vrstiev
- *C* je podmnožina NxN reprezentujúca množinu orientovaných spojov; tieto spoje vedú len medzi susednými vrstvami a každé 2 po sebe nasledujúce vrstvy tvoria úplný bipartitný graf
- *I* je podmnožina N reprezentujúca vstupné neuróny
- *O* je podmnožina N reprezentujúca výstupné neuróny
- *w*: *C*→***R*** je váhová funkcia
- t: *N*→***R*** je prahová funkcia

Nech máme neurón *X* s *n* vstupmi a váhami *($w_1,...,w_n$)*, prah ϑ, vstup $\vec{z} = (z_1,...,z_n)$ a prenosovú funkciu *f*:

$$f(\xi) = \frac{1}{1+e^{-\xi}}$$

potom výstup neurónu *X* označíme *y* a vypočítame ho:

$$y = f\left(\sum_{i=1}^{n} w_i z_i + \vartheta\right)$$

Nech neurónová sieť má *m* neurónov vo výstupnej vrstve. Potom výstupom tejto siete je m-tica $\vec{y} = (y_1,...,y_m) \in R^m$, kde y_j je výstupom j-tého neurónu výstupnej vrstvy.

Aj keď analytický model tvorený neurónovou sieťou nie je tak transparentný ako rozhodovací strom a neumožňuje export vo forme rozhodovacích pravidiel, môže byť veľmi dobrým klasifikačným modelom. Jeho kvalitu určuje hlavne zvolený postup učenia ako aj kvalita trénovacích dát. Najznámejšia forma učenia neurónovej siete je učenie s učiteľom, ktorú používa aj algoritmus spätného šírenia.

Algoritmus spätného šírenia

Algoritmus spätného šírenia (*Back-propagation algorithm*) je algoritmus slúžiaci na učenie vrstevnatých neurónových sietí. Algoritmus prvý krát popísal Paul Weber v roku 1974, avšak do pozornosti sa dostal až po výskume Davida E. Rumelharta, Goeffrey E. Hintona a Ronalda J. Williamsa v roku 1986. Jedná sa o iteratívnu gradientnú metódu, ktorá adaptuje synaptické váhy s cieľom minimalizovať strednú kvadratickú odchýlku medzi očakávaným a skutočným výstupom vrstevnatej neurónovej siete. Ako prenosová funkcia sa používa hladká nelineárna funkcia, najčastejšie sigmoidálna funkcia. Hladká funkcia je totiž diferencovateľná, čo je u vrstevnatých neurónových sietí kľúčová vlastnosť pre adaptáciu váh v procese učenia [5]. Typicky sa volí rovnaká prenosová funkcia pre celú sieť.

Pred zahájením učenia musíme nadefinovať topológiu neurónovej siete – počet vrstiev a počty neurónov na jednotlivých vrstvách. Podľa [5] platí, že s rastúcim počtom skrytých vrstiev a neurónov v nich sa zvyšujú časové aj dátové nároky na proces učenia siete. Veľká sieť má tiež tendenciu preučiť sa. Ako už bolo spomenuté, autori publikácie [3] odporúčajú práve jednu skrytú vrstvu. V [5] sa uvádza heuristika, podľa ktorej by počet neurónov v skrytej vrstve mal byť dvakrát väčší, ako vo vrstve vstupnej.

Atribúty pre vstupné vzory sú spravidla za účelom urýchlenia učiacej fázy normalizované tak, aby spadli do intervalu [0,1]. V prípade spojitých atribútov je možné použiť napríklad min-max normalizáciu. Kategoriálne atribúty sa spravidla zakódujú tak, že pre každú kategóriu atribútu sa vytvorí vstupný neurón. Pre každý záznam sa v závislosti od kategórie daného atribútu aktivuje príslušný neurón. Neuróny odpovedajúce ostatným kategóriám majú pre tento záznam na vstupe 0. Výstupná vrstva sa zariadi analogicky.

Algoritmus spätného šírenia pozostáva zo 6tich krokov:

1. ***Vytvorenie siete***. Zvolíme počet vrstiev a počet neurónov v nich; zvolíme prenosovú funkciu a učiacu konštantu.
2. ***Inicializácia váh***. Zvolíme hodnoty synaptických váh ako malé náhodné hodnoty.
3. ***Predloženie vstupného vzoru***. Sieti predložíme vstupný vzor $\vec{x} = (x_1,...,x_n)$ a požadovaný výstup $\vec{d} = (d_1,...,d_m)$.
4. ***Výpočet skutočného výstupu siete***. V tomto kroku postupujeme od vstupnej vrstvy k výstupnej a počítame hodnoty neurónov:

$$y_j = f\left(\sum_{i=1}^{n} x_i w_{ij} + \vartheta_j\right)$$

5. ***Určenie chyby***. Z kroku 4 sme dostali výstup $\vec{y} = (y_1,...,y_m)$. Spočítame hodnotu chybovej funkcie oproti očakávanému výstupu pre tento záznam $\vec{d} = (d_1,...,d_m)$:

$$E = \frac{1}{2}\sum_p \sum_j \left(y_{jp} - d_{jp}\right)^2$$

   Ak je chyba dostatočne malá (akceptovateľnú hodnotu zvolíme dopredu), alebo bol dosiahnutý maximálny počet cyklov, tak skončíme.

6. ***Adaptácia synaptických váh***. V smere od výstupnej vrstvy k vstupnej adaptujeme váhy v závislosti od toho, či je neurón vo výstupnej alebo skrytej vrstve:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha\delta_j y_i ,$$

   kde *t* reprezentuje jednotlivé iterácie (čas), α reprezentuje učiacu konštantu, ktorá nadobúda hodnôt z intervalu (0,1),

$$\delta_j = y_j(1-y_j)(d_j - y_j)$$

   pre výstupnú vrstvu a

$$\delta_j = y_j(1-y_j)\sum_k \delta_k w_{jk}$$

   pre skrytú, kde *k* je index pre vrstvu nad *j*.

7. Prejdi na krok 3.

4.3. Metóda k-najbližších susedov

Táto metóda je založená na hľadaní najbližších susedov a klasifikácii na základe hodnoty ich cieľových atribútov. Samotný model je v podstate tvorený trénovacou množinou a proces učenia spočíva v uschovaní príkladov v modeli. Vstupné atribúty musia byť numerické, aby bolo možné pre každé 2 príklady vypočítať ich vzájomnú vzdialenosť. V opačnom prípade je potrebné pri predspracovaní dát vykonať odpovedajúce transformácie.

Ak je cieľový atribút spojitý, určí sa hodnota cieľového atribútu pre nový záznam ako priemer z hodnôt *k* najbližších susedov v modeli. Ak je cieľový atribút kategoriálny, klasifikuje sa záznam na základe väčšinového výberu (hlasovania) susedov. Predpokladám, že lepšie výsledky by mohli byť dosiahnuté váženým hlasovaním resp. váženým priemerom vzhľadom k vzdialenosti susedov.

Každý spracovávaný vzor reprezentuje bod v n-rozmernom príznakovom priestore. Dimenzie tohto priestoru odpovedajú jednotlivým príznakom vstupných vzorov. Pre dosiahnutie lepších výsledkov je potrebné hodnoty jednotlivých atribútov normalizovať, napríklad do intervalu [0,1].

Výpočet vzdialenosti

Dôležité je zvoliť metriku, v ktorej sa bude počítať vzdialenosť objektov (vzorov). Vo väčšine prípadov sa používa Euklidovská metrika, ktorá pre 2 vzory $\vec{x} = (x_1,...,x_n)$ a $\vec{y} = (y_1,...,y_n)$ definuje ich vzdialenosť v n-rozmernom priestore ako:

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} .$$

Algoritmus

Myšlienka použiť najbližších susedov na rozpoznávanie vzorov bola navrhnutá v roku 1951 (Fix, Hodges). Nasledovná podoba algoritmu bola uverejnená v roku 1997 Mitchellom a predpokladá, že cieľový atribút je kategoriálny:

1. ***Učenie***

   1.1. Každý trénovací záznam $[\vec{x}, y]$ priraď do bázy záznamov.

2. ***Klasifikácia***

   2.1. Vezmi nový záznam $\vec{x} = (x_1,...,x_n)$.

   2.2. Nájdi $\vec{x}_1,...,\vec{x}_k$ najbližších záznamov v báze záznamov.

   2.3. Priraď mu klasifikáciu *y*, kde $\delta(y_i, y_l) = 1$ pre $y_i = y_l$, inak 0:

$$y = y_j \Leftrightarrow \sum_{l=1}^{k} \delta(y_j, y_l) = \max_i \sum_{l=1}^{k} \delta(y_i, y_l)$$

Porovnanie s ostatnými

Klasifikátor založený na metóde k-najbližších susedov sa zaraďuje medzi klasifikátory s ***lenivým učiteľom***. Na rozdiel od rozhodovacích stromov alebo neurónových sietí, kde pri učení vytvoríme komplexný klasifikačný model, je v tomto prípade uchovávaná celá trénovacia vzorka dát, ktorá je pri klasifikácii používaná na hľadanie susedov. Pri každej klasifikácii sa znovu vypočítava vzdialenosť, čo značne spomaľuje tento proces a tiež kladie nemalé kapacitné nároky na uschovanie modelu.

4.4. Analýza nákupného košíka

Rozdiel metódy MBA (*Market Basket Analysis*) od predošlých metód je hlavne v tom, že jej výsledkom nie sú rozhodovacie (klasifikačné) pravidlá, ale ***asociačné pravidlá***. Asociačné pravidlá sú odlišné v tom, že žiadny z atribútov nie je označený ako cieľový a pravidlo tak vyjadruje závislosti medzi nimi. Typickým asociačným pravidlom by mohlo byť pravidlo: {maslo, chlieb} → {mlieko}.

V prípade analýzy nákupného košíka nie je táto definícia presná. MBA predpokladá kategoriálne dáta, ktoré odpovedajú určitým transakciám. Počet hodnôt v jednom zázname môže byť preto premenlivý, jednotlivé hodnoty nie sú usporiadané a neodpovedajú hodnotám atribútov, ale len položkám transakcie. Pravidlá preto majú tvar ***X***→***Y***, kde *X* a *Y* sú množiny obsahujúce položky z týchto transakcií.

Názov tejto metódy napovedá, že ideálnym príkladom dát sú nákupné účtenky a pravidlá vyjadrujú závislosti medzi skupinami kupovaných položiek. Výsledným pravidlom by v tomto prípade mohlo byť: chlieb→maslo. Toto pravidlo nazývame ***triviálne***, pretože obe množiny (strany pravidla) obsahujú práve jednu položku.

Hodnotenie pravidiel

Na výber tých najlepších pravidiel sú stanovené 3 základné kritéria hodnotenia. Nech *M* je veľkosť množiny všetkých položiek a nech máme:

- $I = \{i_1, ..., i_M\}$ množinu všetkých položiek
- $T = (t_1, ..., t_N),\ t_i \subseteq I$ je množina transakcií
- pravidlo $X \to Y,\ kde\ X \subset I, Y \subset I, X \cap Y = \varnothing$
- zobrazenie *n*, ktoré priradí množine položiek počet transakcií, ktoré tieto položky obsahujú

Prvé kritérium sa nazýva ***podpora pravidla***. Toto kritérium určuje aké veľké má toto pravidlo zastúpenie, teda koľko transakcií percentuálne obsahuje hodnoty z množiny *X* a zároveň *Y*:

$$Support(X \rightarrow Y) = \frac{n(X \cup Y)}{N} \bullet 100\%$$

Druhým kritériom je ***spoľahlivosť pravidla***. Toto kritérium určuje mieru relevantnosti pravidla v percentách. Jeho hodnota nám ukazuje, ako často sa vyskytujú v transakcii položky z množiny *Y*, ak sa tam vyskytujú položky z množiny *X*:

$$Confidence(X \rightarrow Y) = \frac{n(X \cup Y)}{n(X)} \bullet 100\%$$

Tretím kritériom je ***zlepšenie***. Veľkosť hodnoty tohto kritéria odpovedá miere zlepšenia pri použití tohto pravidla oproti náhodnej voľbe:

$$Lift(X \rightarrow Y) = \frac{p(X \cup Y)}{p(X)p(Y)} = \frac{\frac{n(X \cup Y)}{N}}{\frac{n(X)}{N} \times \frac{n(Y)}{N}} = \frac{n(X \cup Y)}{n(X)n(Y)} \times N$$

Ak je jeho hodnota menšia ako 1, je použitie tohto pravidla horšie ako náhodná voľba.

Algoritmus APRIORI

Publikácia [5] uvádza základnú verziu algoritmu, ktorý R. Agrawal navrhol v súvislosti s analýzou nákupného košíka (Agrawal a kol, 1996). Uľahčuje a zrýchľuje generovanie pravidiel tým, že hľadá frekventované množiny položiek. Z týchto sa potom generujú samotné pravidlá. Množina položiek $X=\{x_1,...,x_k\}$ sa nazýva ***frekventovaná***, ak jej podpora je väčšia, ako minimálna stanovená podpora *minSupport*. Podporu sme definovali pre pravidlo. Pre množinu je definícia analogická:

$$Support(X) = \frac{n(X)}{N}$$

Algoritmus využíva „apriori“ vlastnosť frekventovaných množín: Všetky neprázdne podmnožiny frekventovanej množiny musia byť frekventované. To ovplyvňuje spôsob, akým sa frekventované množiny generujú. Na začiatku sa vyberú frekventované položky (množiny položiek veľkosti 1). Je zrejmé, že ak nejaká množina veľkosti 1 (položka) nemá požadovanú podporu, tak každá nadmnožina takejto množiny má podporu menšiu alebo rovnú podpore

pôvodnej množiny. Z frekventovaných množín veľkosti *k* sa vytvárajú spájaním frekventované množiny veľkosti *k+1*.

Nech L_k označuje množinu frekventovaných množín položiek (kombinácií) s *k* prvkami. Spájanie prebieha nasledovne:

Apriori algoritmus

1. do L_1 daj položky, ktoré spĺňajú minimálnu požadovanú podporu
2. zvoľ $k=2$
3. kým L_{k-1} obsahuje nejaké prvky opakuj
   3.1. z L_{k-1} vytvor množinu kandidátov C_k pomocou funkcie ***apriori-gen***
   3.2. z C_k vytvor L_k filtrovaním kombinácií, ktoré nedosiahli požadovanú minimálnu podporu
   3.3. zväčši *k* o 1
4. vráť zjednotenie všetkých L_k

Apriori-gen (L_{k-1})

1. pre všetky dvojice kombinácii $Comb_1$, $Comb_2$ z L_{k-1}
   1.1. ak sa $Comb_1$, $Comb_2$ zhodujú v práve *k-2* položkách, pridaj ich zjednotenie do množiny kandidátov C_k
2. pre každú kombináciu *Comb* z C_k
   2.1. ak niektorá z podkombinácii dĺžky *k-1* nie je obsiahnutá v L_{k-1}, odstráň *Comb* z C_k

Získané frekventované množiny sa použijú na generovanie pravidiel. V tomto prípade sa už pozerá hlavne na spoľahlivosť a zlepšenie. Jednotlivé frekventované množiny *X* rozdeľujeme na 2 disjunktné podmnožiny, ktorých zjednotenie tvorí *X*, teda

$$R, L \subset X : \; R \cap L = \varnothing \quad \wedge \quad R \cup L = X \;,$$

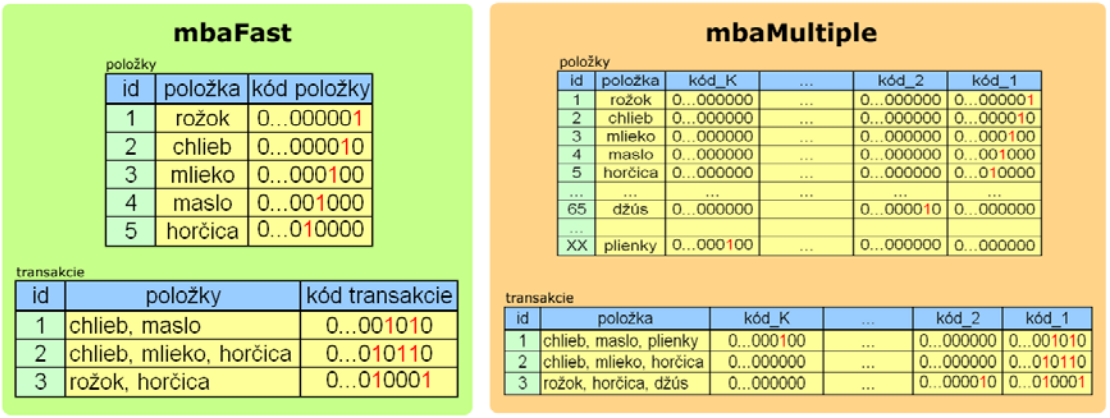
a testujeme, či pravidlá $L \rightarrow R$ spĺňajú požiadavku na minimálnu spoľahlivosť a zlepšenie.

4.5. Vlastná implementácia MBA

Implementácia analýzy nákupného košíka a algoritmu Apriori má podobu databázovej procedúry. Uprednostnil som tento variant pred klasickou aplikačnou implementáciou z viacerých dôvodov. Hlavným z nich je fakt, že databázové systémy sú optimalizované na operácie nad veľkými množinami dát, ktoré sa v algoritme často vyskytujú. Oproti „kurzorovitému“ aplikačnému spracovaniu (kedy musíme záznamy spracovávať po jednom) sme schopný výsledky získať v kratšom čase. Ďalším dôvodom je šetrenie komunikačnej linky v prípade, že dáta sa nachádzajú na vzdialenom serveri ako aplikácia.

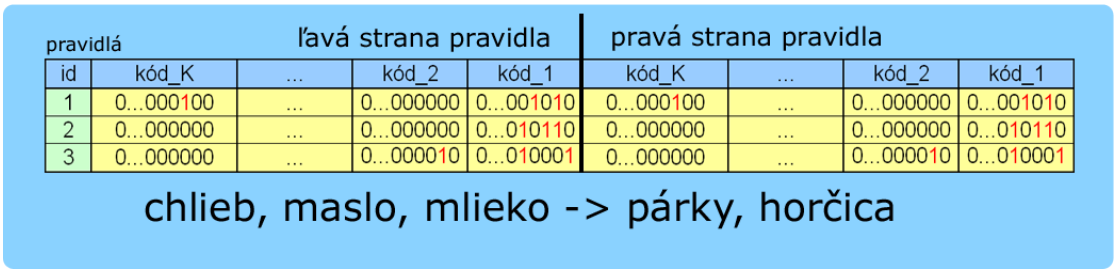
Hlavná idea spočíva v tom, že sa položkám priradí unikátny ***kód***, ktorým je kladné celé číslo. Tento sa volí tak, aby jeho binárna reprezentácia

obsahovala práve jednu jednotku a tento kód bol unikátny medzi všetkými položkami. Nech *P* je položka s poradovým číslo *p*. Potom sa jej priradí kód 2^{p-1}. Zlepšenie spočíva v tom, že sa obdobne zakódujú aj transakcie a pravidlá. Potom kód každej transakcie a pravidla tvorí bitovú masku všetkých prítomných položiek. Na získanie počtu transakcií s nejakou množinou položiek sa používa binárny operátor AND. Toto sa javí ako rýchlejší prístup, než prechádzať všetky transakcie a overovať prítomnosti jednotlivých položiek.



obr. 10 – Porovnanie verzií algoritmu mbaFast a mbaMultiple

Problémom je, že dátový typ *BIGINT* pre MS SQL Server má 8 bytov (64 bitov), pričom jeden bit je použitý na znamienko. Preto môžeme zakódovať do jedného takéhoto poľa len 63 položiek. Toto obmedzenie má rýchla verzia algoritmu ***mbaFast***. Ak je položiek viac, vytvorí sa odpovedajúci počet kódových polí, ktoré sa logicky radia za seba (prvé reprezentuje 63 najmenej významných bitov a podobne) a umožňujú zakódovať ľubovoľný počet položiek až do maximálneho počtu 1020×63[10]. Takto funguje verzia algoritmu ***mbaMultiple***, ktorá je trošku pomalšia ako *mbaFast*, pretože kvôli premenlivému počtu kódovacích polí sa časť kódu generuje dynamicky. Porovnanie, ako jednotlivé verzie algoritmu pristupujú k položkám a transakciám, vidieť na obr. 10.



obr. 11 – Interná reprezentácia pravidla v prípade verzie mbaMultiple

[10] Toto obmedzenie vyplýva z faktu, že tabuľka v MS SQL Server Express 2005 môže mať maximálne 1024 stĺpcov, pričom maximálne 4 stĺpce sú v pomocných tabuľkách použité na uchovanie ostatných informácií.

V prípade pravidiel musí byť počet kódovacích polí dvojnásobný – pre ľavú aj pravú stranu pravidla. Internú reprezentáciu pravidla vidíme na obr. 11. Samotný algoritmus *Apriori* som mierne modifikoval. Stručný zápis použitého algoritmu v pseudokóde vyzerá nasledovne:

1. ***normalizuj dáta*** – vytvor tabuľku transakcií *mba_trans* a tabuľku položiek *mba_items* (tabuľka *mba_trans* má dvojice *[id_transakcie, id_položky]*)
2. ***zakóduj položky*** – do tabuľky položiek pridaj potrebný počet stĺpcov pre kódy a zakóduj položky
3. ***zakóduj transakcie*** – podľa kódov položiek zakóduj transakcie (tabuľka *mbam_trans_code*)
4. ***vygeneruj triviálne pravidlá*** – vytvor pravidlá s 2 položkami a spočítaj pre ne podporu, spoľahlivosť a zlepšenie
5. ***vytvor novú generáciu*** – pravidlá, ktoré spĺňajú požiadavku na minimálnu podporu, skopíruj do tabuľky pravidiel pre novú generáciu (tabuľka *mbam_rules_newGen*)
6. ***cyklus generovania netriviálnych pravidiel*** – kým je tabuľka novej generácie neprázdna, opakuj
   6.1. skopíruj pravidlá do tabuľky pravidiel (*mbam_rules_ext*)
   6.2. kombinuj pravidlá podľa ľavých a pravých strán s triviálnymi a vytvor kandidátov na novú generáciu (tabuľka *mbam_rules_temp*)
   6.3. spočítaj podporu, spoľahlivosť a zlepšenie pre tieto pravidlá
   6.4. skopíruj pravidlá splňujúce minimálnu podporu do tabuľky pre novú generáciu *mbam_rules_newGen*
7. ***vytvor výstup*** – rozkóduj pravidlá späť na položky a vytvor tabuľku s výstupnými pravidlami (tabuľka *mbam_output*)

Generovanie pravidiel sa oproti algoritmu *Apriori* mierne líši. Platí však, že každé pravidlo možno vytvoriť kombináciou triviálnych pravidiel. To nám zaručuje, že sme žiadne pravidlo nevynechali. Na druhej strane je tento prístup dostatočne rýchly, keďže sa kombinujú vždy len frekventované pravidlá.

V mojej implementácii rozlišujem 2 hodnoty minimálnej podpory *minSup1* a *minSup2*. Jedna sa vzťahuje na triviálne pravidlá, druhá na komplexnejšie. Dôvodom je, že komplexné pravidlá môžu mať pre nás veľkú hodnotu, ale s komplexnosťou pravidla klesá jeho podpora. MBA sa spúšťa zavolaním DB procedúry ***sp_mbaMain***, ktorá má ako parametre:

- Názov tabuľky, ktorá obsahuje trénovacie/testovacie dáta
- Minimálna podpora pre triviálne pravidlá
- Minimálna podpora pre komplexné pravidlá
- Minimálna spoľahlivosť
- Minimálne zlepšenie

Táto procedúra zistí počet položiek a zavolá príslušnú verziu algoritmu. Príklad volania tejto databázovej procedúry vyzerá nasledovne:

```
execute sp_mbaMain 'my_data', 0.1, 0.05, 0.5, 1
```

Klasifikácia pomocou MBA

Analýza nákupného košíka nie je metódou dobývania znalostí vytvorenou a vhodnou na klasifikáciu. Zistil som, že po malej modifikácii, by nám ale táto metóda mohla poslúžiť ako klasifikátor.

Nech máme dáta, ktoré majú vstupné atribúty $A_1,...,A_n$ a C ako cieľový atribút. Kategorizujeme spojité atribúty a upravíme (premenujeme) hodnoty všetkých atribútov tak, aby ich domény boli disjunktné množiny. V tomto prípade sa jednotlivé hodnoty premenujú do tvaru: „*atribút=hodnota*“, čo za predpokladu, že názvy atribútov ani hodnoty neobsahujú znak „=“, zaručuje unikátnosť. Tieto hodnoty sa použijú ako položky v analýze nákupného košíka. Budeme uvažovať len pravidlá, ktoré majú tvar $\boldsymbol{X \rightarrow y}$, kde X je množina hodnôt vstupných atribútov a y je hodnota cieľového atribútu. Vylúčiť môžeme tiež pravidlá, ktoré obsahujú 2 hodnoty z toho istého atribútu, pretože takéto pravidlá sú protirečivé.

Nech k_i je veľkosť domény atribútu A_i a nech k je veľkosť domény cieľovej premennej. Potom existuje $(k_1+1)(k_2+1)\ldots(k_n+1)k$ pravidiel, ktoré spĺňajú tieto podmienky. Jedná sa o všetky možné kombinácie, pričom u vstupných atribútov musíme pripočítať k veľkosti domény jednotku, pretože môžeme tento atribút v pravidle úplne vynechať (neuvedieme žiadnu hodnotu). Hodnota cieľového atribútu musí byť prítomná v každom pravidle.

Keď už máme výsledné pravidlá, klasifikácia prebieha nasledovne. Prechádzame cez pravidlá, ktoré sú zoradené podľa spoľahlivosti. Keď nájdeme prvé, ktoré vyhovuje príkladu, tak ho použijeme na klasifikáciu. Druhým variantom je možnosť použiť vážené hlasovanie, pričom váhu hlasu určí spoľahlivosť a podpora pravidla.

Verzia algoritmu analýzy nákupného košíka pre klasifikáciu sa volá ***mbaClass***. Tento algoritmus sa spúšťa zavolaním DB procedúry ***sp_mbaClassMain***, ktorá má ako parametre:

- Názov tabuľky, ktorá obsahuje trénovacie/testovacie dáta
- Názov cieľového atribútu
- Názov atribútu, ktorý obsahuje ID každého záznamu
- Minimálna podpora pre triviálne pravidlá
- Minimálna podpora pre komplexné pravidlá
- Minimálna spoľahlivosť
- Minimálne zlepšenie
- Percento (náhodne vybraných) trénovacích dát
- Percento (náhodne vybraných zo zvyšku) dát na testovanie

Algoritmus sa líši oproti neklasifikačnej verzii v niekoľkých drobnostiach. Dáta sa v úvode delia na trénovacie a testovacie. Hodnoty pre cieľový atribút sa nekódujú. Tým pádom tabuľky generovaných pravidiel obsahujú kódy položiek len pre ľavú stranu pravidiel. Na druhej strane obsahujú navyše jeden stĺpec - bitovú masku *code_columns*. Tento stĺpec určuje, ktoré atribúty už v pravidle figurujú, aby nedošlo pri generovaní k vytvoreniu protirečivého pravidla. V závere treba vykonať testovanie modelu.

4.6. Porovnanie metód

V tejto kapitole porovnám metódy na dobývanie znalostí dát popísané v predošlých kapitolách. Na testovanie metód použijem umelé dáta, ktoré boli vygenerované databázovou procedúrou, aby overili správnosť a rýchlosť algoritmov. Tabuľky obsahujúce umelé dáta sú súčasťou CD a ich import je podrobne popísaný v kapitole 7.8. Spustením skriptu sa vytvoria nasledovné tabuľky:

- ***data_sample_knn*** – dáta, ktoré som použil na testovanie algoritmu k-najbližších susedov a porovnanie rýchlosti jeho databázovej a aplikačnej verzie.
- ***data_sample_knn_cont*** – dáta, ktoré som použil na testovanie algoritmu k-najbližších susedov na predikciu (spojitý cieľový atribút).
- ***data_sample_c45*** – dáta, ktoré som použil na testovanie algoritmu C4.5.
- ***data_sample_c45_2*** – dáta, ktoré som použil na testovanie algoritmu C4.5 (obsahujú viac atribútov ako predošlá tabuľka).
- ***data_sample_classif*** – dáta, ktoré som použil na testovanie algoritmov ID3, klasifikačnej verzie analýzy nákupného košíka a algoritmu C4.5.

Test algoritmov

Každý algoritmus som podrobil 2 testom. V prvom teste bola použitá celá dátová množina, ktorá obsahovala 10000 riadkov. Druhý test bežal na 500 riadkoch. Na testovanie som použil moju aplikáciu, pričom som využil možnosť testovať algoritmus pre rôzne veľkosti a pre každú veľkosť som test opakoval 4 krát. Túto funkčnosť som podrobne popísal v treťom odseku kapitoly 7.6.

U jednotlivých algoritmov som sledoval celkový čas učenia vrátane klasifikácie, priemernú celkovú správnosť modelu, časovú krivku a učiacu krivku. Výsledky som zobrazil v nasledujúcej tabuľke:

		ID3	C4.5	MBAC	KNN	KNN appl
Typ modelu		rozhodovací strom	rozhodovací strom	pravidlá	trénovacie dáta	trénovacie dáta
Čas	učenie	minúty až hodiny	minúty až hodiny	minúty až hodiny	sekundy	sekundy
	testovanie	sekundy	sekundy	sekundy	minúty	minúty
Atribúty	vstupné	diskrétne	diskrétne / spojité	diskrétne	numerické (spojité, diskrétne)	numerické (spojité, diskrétne)
	cieľový	diskrétny	diskrétny	diskrétny	diskrétny / spojitý	diskrétny
Test 1	čas	0:08:37	0:08:47	0:21:42	2:12:00	3:21:18
	celková správnosť	99,46%	100,00%	94,63%	71,57%	68,64%
Test 2	čas	0:01:08	0:01:10	0:03:57	0:00:35	0:00:48
	celková správnosť	98,14%	98,41%	94,56%	72,78%	70,97%

obr. 12 - Porovnanie algoritmov na dobývanie znalostí z dát

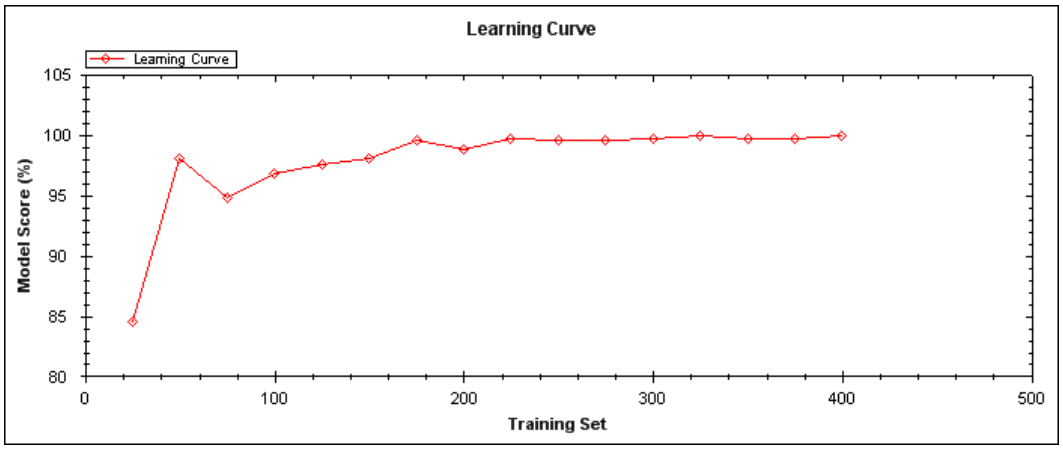
V tabuľke vidíme, aký typ modelu dostaneme použitím daného algoritmu, aký je očakávaný čas učenia a klasifikácie a s akým typom atribútov si dokáže algoritmus poradiť. V spodnej časti tabuľky sú zaznamenané výsledky testu.

Vidíme, že obidve implementácie algoritmu k-najbližších susedov – aplikačná (KNN appl) aj databázová (KNN) – sú pri veľkom množstve dát značne pomalé. Dôvodom je veľká časová zložitosť klasifikácie, pri ktorej sa pre každý vstupný záznam musí spočítať vzdialenosť ku všetkých záznamom v trénovacej množine. Na druhej strane pri malom množstve dát boli skoro 2x rýchlejšie. Bohužiaľ celková správnosť je vzhľadom k faktu, že boli použité umelé dáta, veľmi nízka.

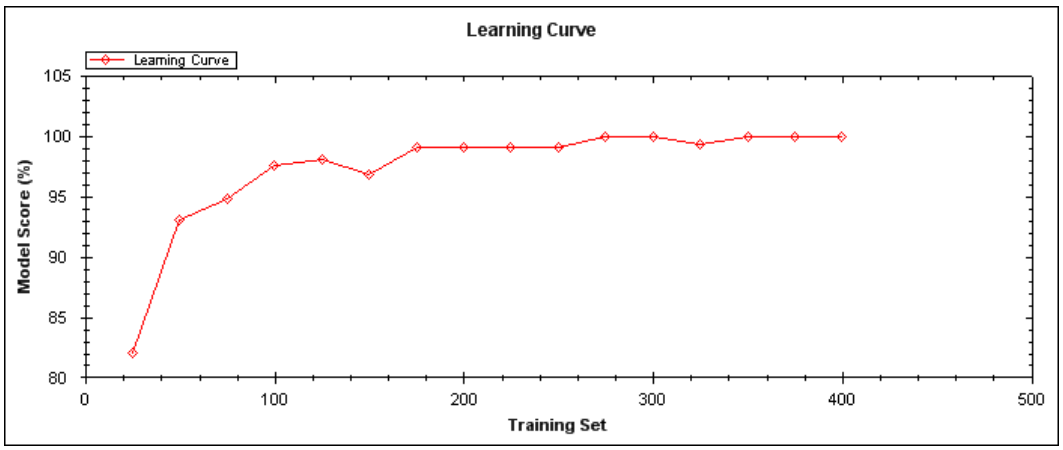
Algoritmy ID3 a C4.5 generujúce rozhodovacie stromy sa ukázali ako veľmi rýchle a spoľahlivé na väčšej aj menšej trénovacej množine. Časy učenia vrátane klasifikácie sú tiež viac ako uspokojivé. Klasifikačná verzia algoritmu MBA (MBAC) potrebuje zhruba 3x viac času ako algoritmy pre rozhodovacie stromy. Na moje prekvapenie však tento algoritmus vykazuje veľmi dobrú priemernú celkovú správnosť.

Učiace krivky

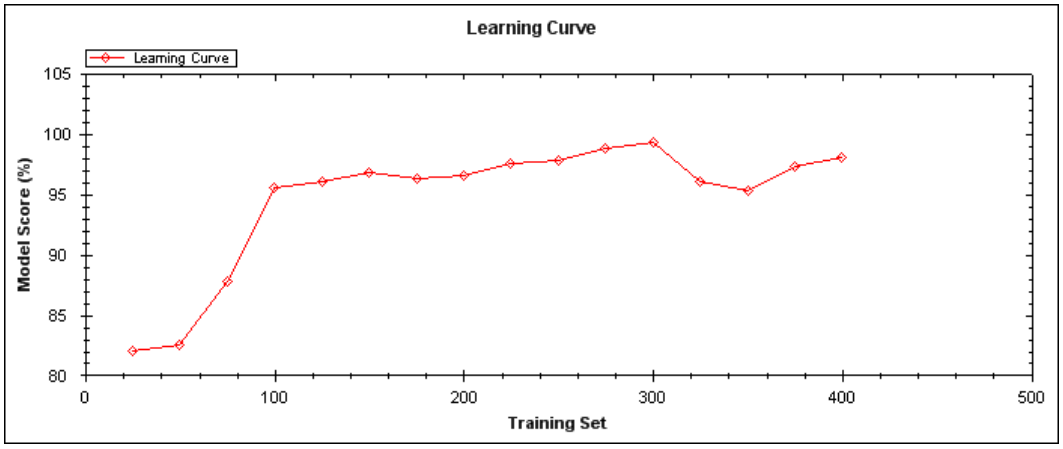
Učiace krivky dosiahnuté pri učení môžeme vidieť na obr. 13 – obr. 17. U algoritmov ID3, C4.5 a MBAC vidieť nárast skóre modelu s narastajúcou veľkosťou trénovacej množiny. Pre obidve implementácie algoritmu k-najbližších susedov však celková správnosť modelu s narastajúcou veľkosťou trénovacích dát kolíše.



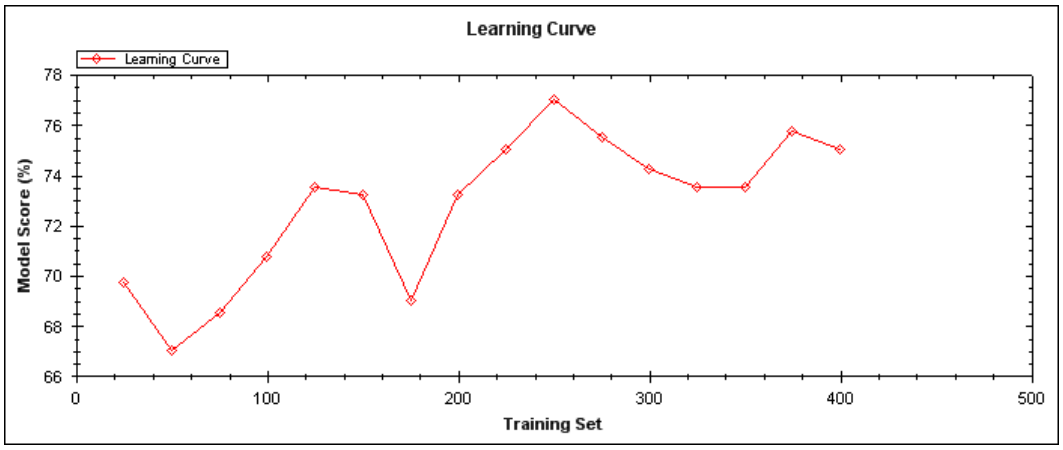
obr. 13 – Učiaca krivka pre algoritmus ID3



obr. 14 – Učiaca krivka pre algoritmus C4.5



obr. 15 – Učiaca krivka pre algoritmus MBAC



obr. 16 – Učiaca krivka pre databázovú verziu algoritmu k-najbližších susedov

obr. 17 – Učiaca krivka pre aplikačnú verziu algoritmu k-najbližších susedov

Porovnanie databázového a aplikačného prístupu

V úvode práce som vyslovil predpoklad, že algoritmy na dobývanie znalostí z dát implementované formou uložených databázových procedúr budú rýchlejšie ako aplikačné implementácie. Túto úvahu podrobnejšie rozoberám v kapitole 7.1 na strane 59. Túto hypotézu som sa pokúsil potvrdiť implementáciou zhodného algoritmu (algoritmu k-najbližších susedov) v oboch variantoch.

V tabuľke výsledkov na obr. 12 vidíme porovnanie rýchlosti na malej aj veľkej trénovacej množine. Z výsledkov vidíme, že implementácia formou databázovej procedúry je zhruba 1,5 – krát rýchlejšia. To potvrdilo moju hypotézu. Na druhej strane som však predpokladal, že tento koeficient bude vyšší. Lepšie výsledky by mohli byť dosiahnuté výkonnejším databázovým systémom.

Záver

Dáta aj zadanie úlohy vyžaduje pri prvej analýze použitie algoritmu analýzy nákupného košíka. Preto nebolo potrebné túto voľbu podrobovať

žiadnym testom. Použijem vylepšenú databázovú verziu klasického algoritmu, ktorú som navrhol a detailne popísal na strane 41.

Druhá úloha vyžaduje tvorbu klasifikačného modelu. Na základe predošlých výsledkov sa ukazuje ako najlepšia voľba niektorý z dvoch rozhodovacích stromov. Napriek tomu sa pokúsim aplikovať aj vlastný algoritmus MBAC, ktorý za rozhodovacími stromami v testoch mierne zaostáva. Ako absolútne nevhodný sa ukázal pre moju analýzu algoritmus k-najbližších susedov. Tento algoritmus vykazoval slabú schopnosť správnej klasifikácie a na väčšej množine dát aj veľké časové nároky. Keďže dáta pre druhú analýzu sú pomerne obsiahle, mohlo by to celý proces dobývania znalostí značne spomaliť.

5. Analýza 1 – Mäsokombinát

5.1. Popis úlohy a dát

Úlohou je metódou analýzy nákupného košíka nájsť závislosti medzi kupovanými mäsovými výrobkami. Dáta boli dodané v podobe exportu z databázového systému *Oracle 10g*. Jednalo sa však o úplný export databázy, ktorá obsahovala stovky tabuliek, takže dodaný súbor mal viac ako 23 GB.

Najskôr bolo potrebné identifikovať, ktoré dátové tabuľky obsahujú požadované vstupné dáta. K danej databáze neexistovala žiadna dokumentácia a názvy tabuliek a ich stĺpcov boli nezmysluplné. Použitím modelovacieho nástroja *Erwin* sa mi podarilo získať ER diagram databázy (funkcia *Reverse Engineering*). ER diagram (*Entity-Relationship diagram*) je diagram, ktorý zobrazuje všetky tabuľky v databáze a ich vzájomné vzťahy.

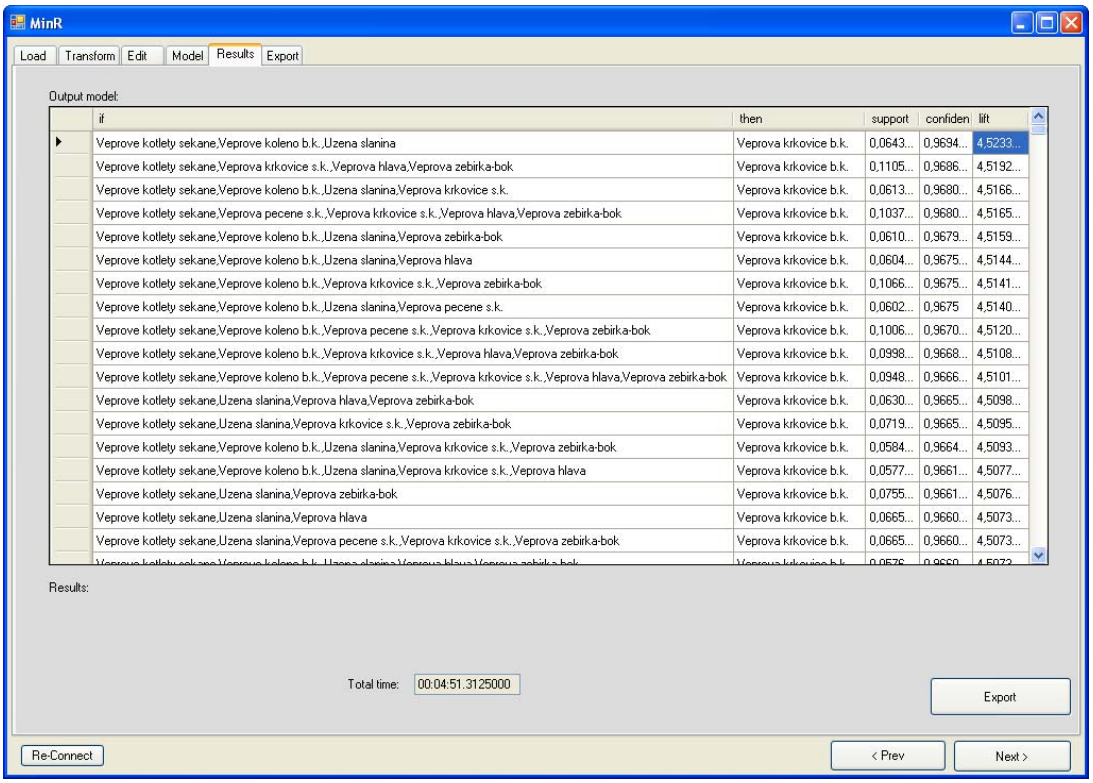
Náhľadom na dáta sa mi podarilo nájsť číselník produktov, ktorý bol vzťahom zviazaný s tabuľkou faktúr. Tieto tabuľky mi poslúžili ako zdroj dát, pričom každá faktúra bude reprezentovať jeden nákupný košík.

5.2. Predspracovanie dát

Použitím databázovej procedúry sa mi podarilo dáta pretransformovať do podoby transakcií a exportovať ich do súboru ***data_analysis02.csv***. Keďže analýza nákupného košíka pracuje priamo s položkami jednotlivých transakcií, neboli potrebné žiadne ďalšie transformácie.

5.3. Analýza nákupného košíka

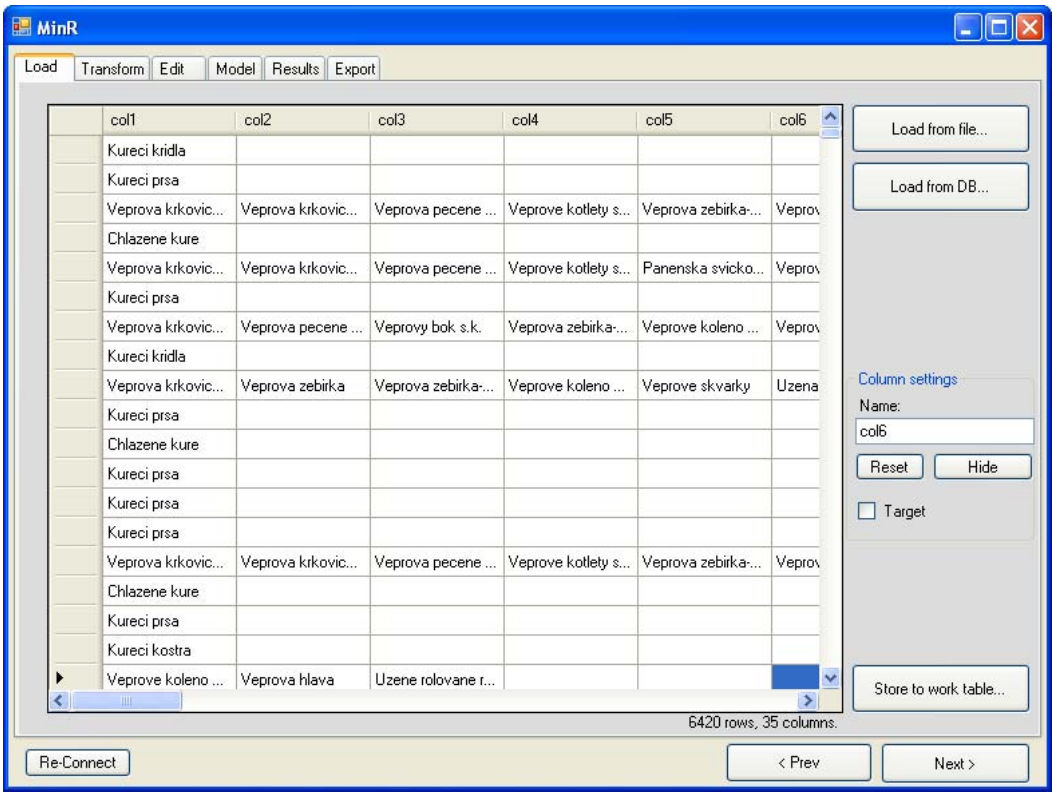
Na získanie pravidiel som použil klasickú verziu analýzy nákupného košíka, ktorú som implementoval vo svojej aplikácii. Popis algoritmu sa nachádza na strane 69. Pri prvom behu som nastavil minimálnu podporu pre triviálne pravidlá na 0.1, pre komplexné pravidlá na 0.05, minimálnu spoľahlivosť pravidla na 0.8 a minimálne zlepšenie na 1. Samotný proces modelovania trval 4 minúty a 51 sekúnd. Výsledné pravidlá zoradené podľa spoľahlivosti a zlepšenia môžeme vidieť na obr. 18.



obr. 18 – Výsledky prvej analýzy

Výsledky modelovania sú prekvapujúce. V prvom rade ma zaskočila maximálna spoľahlivosť, ktorá pri pár pravidlách dosiahla viac ako 96%. Pri takýchto množstvách dát som neočakával tak spoľahlivé a jednoznačné pravidlá. Na druhú stranu sa však všetky pravidlá týkali bravčového mäsa. Na obrázku dokonca vidíme, že pravá strana pravidla je pre najlepších 20 pravidiel zhodná: „Vepřová krkovice bez kosti“.

Prirodzene som sa teda zaujímal, prečo medzi pravidlami nie sú žiadne pravidlá, ktoré sa týkajú hydiny a hovädzieho mäsa. V prípade kuracieho mäsa vysvetľuje jeho neprítomnosť nasledovný obrázok:



obr. 19 – Transakcie

Na obr. 19 vidieť, že transakcie, ktoré obsahujú niektorý z hydinových výrobkov, už neobsahujú žiadnu inú položku. Detailným preskúmaním dát sa mi podarilo zistiť, že zo všetkých transakcií, ktoré obsahujú niektorý z hydinových výrobkov, 96.43% obsahuje práve jednu položku. Takáto informácia by sa dala preložiť do podoby pravidla: „*Ak si kúpim kuracie, už si nekúpim nič iné*“. Z pohľadu zadávateľa bola toto najdôležitejšia znalosť, ktorú sa podarilo odhaliť. Tento príklad ukazuje, ako veľmi je dôležité sa pri procese dobývania znalostí opätovne vracať do predchádzajúcich fáz.

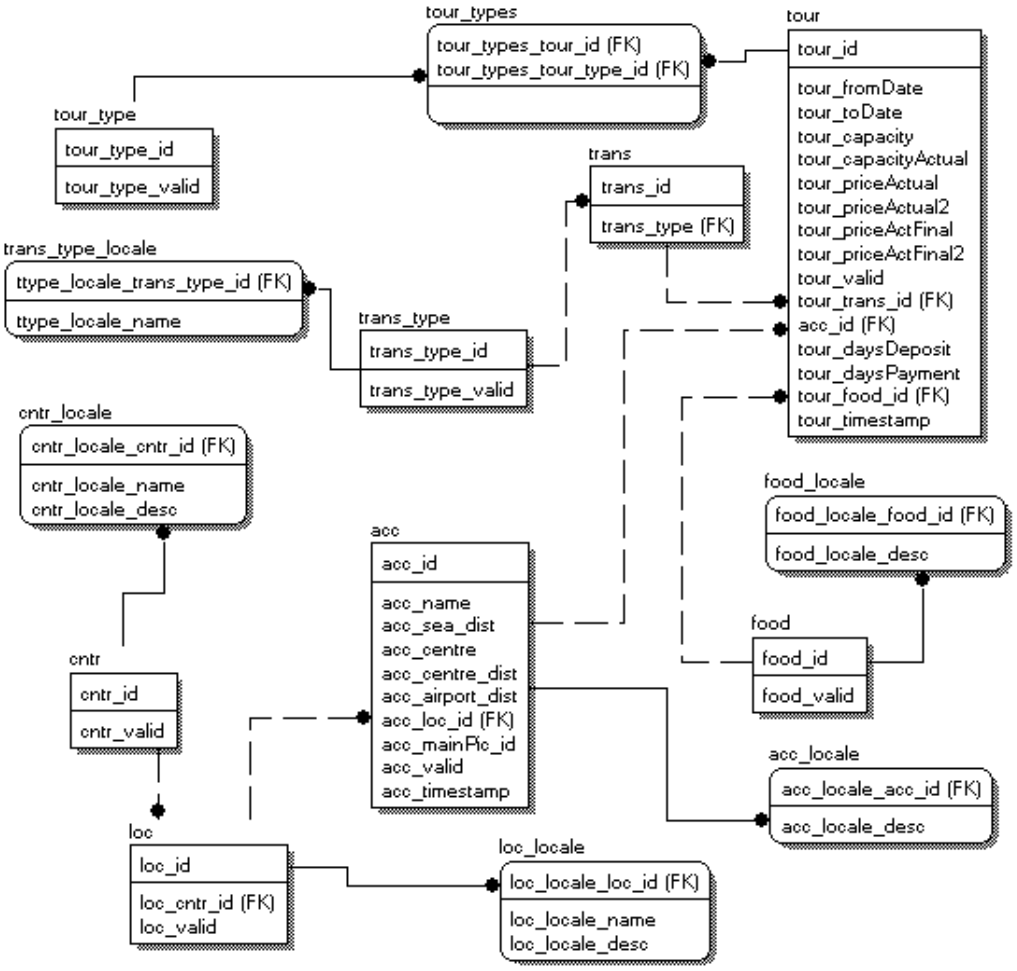
U hovädzieho mäsa bola situácia podobná. Viac ako 68.5% transakcií, ktoré obsahovali niektorý z hovädzích výrobkov už neobsahovali žiadne iné položky. Prekvapivejšie ale bolo zistenie, že menej ako 1% transakcií obsahovalo niektorý z hovädzích produktov. Túto informáciu zadávateľ potvrdil a nebola pre spoločnosť žiadnym prekvapivým zistením.

6. Analýza 2 – Cestovná agentúra

6.1. Popis úlohy a dát

Úlohou je použitím niektorej z klasifikačných metód vytvoriť model, ktorý umožní identifikovať úspešnosť zájazdu v závislosti na jeho atribútoch. Zadávateľ označil za úspešný každý zájazd, v ktorom počet zákazníkov presahuje 50% celkovej kapacity. Ako zdroj dát je použitý export z databázy, ktorá je zdrojom dát pre informačný systém zadávateľa.

Podobne ako pri predošlej analýze, bolo treba najskôr vytipovať potrebné dáta a vytvoriť základný *dataset*[11]. Databáza obsahovala 78 tabuliek, z ktorých sa mi podarilo identifikovať 14, ktoré obsahovali potrebné dáta.



obr. 20 – ER diagram tabuliek použitých pre analýzu

[11] Pod týmto pojmom budeme rozumieť dáta upravené do podoby, v ktorej môžu byť predložené modelovaciemu algoritmu.

Na obr. 20 vidíme ER diagram, ktorý zobrazuje použité tabuľky a vzťahy medzi nimi. Hlavnou tabuľkou je tabuľka *Tour*, ktorá obsahuje základné parametre zájazdu. Vzťahom je spojená s tabuľkou obsahujúcou ubytovacie zariadenia *Acc*, s tabuľkou obsahujúcou rôzne typy stravovania *Food*, s tabuľkou *Trans* obsahujúcou spôsob dopravy a tabuľkou *Tour_type* obsahujúcou typy zájazdov. Každá tabuľka má z dôvodu viacjazyčnosti pôvodného informačného systému sesterskú tabuľku s príponou „*_locale*“, ktorá obsahuje preklady.

6.2. Predspracovanie dát

Pomocou SQL dotazu som spojil tabuľky a získal základný dataset. Tento dataset som exportoval do textového súboru „*data_analysis2_ds1.csv*“, ktorý možno nájsť v adresári „*data*“ na priloženom CD. Ukážku dát vidíme na nasledujúcom obrázku:

	Month	Days	Capacity	Customers	Price	Transport	Locality	Country	Food_Service
1	July	12	70	54	Medium	By plane	Jasna	Slovakia	All Inclusive
2	July	11	90	82	Medium	By plane	Sousse	Tunisia	All Inclusive Club
3	July	11	60	39	Medium	By plane	Hammamet	Tunisia	No food
4	August	12	90	66	Medium	By plane	Costa Brava	Spain	Half-Board
5	August	11	10	6	Low	By plane	Tatranska Lomnica	Slovakia	All Inclusive Light
6	August	11	50	18	High	By plane	Dahab	Egypt	Breakfast
7	August	14	10	4	High	By plane	Bizerta	Turkey	Dinner
8	July	14	70	37	High	By plane	Alanya	Turkey	All Inclusive Pharao
9	August	13	80	76	Medium	By plane	Ibiza	Spain	Full-Board

obr. 21 – Vytvorený dataset

Dataset obsahuje nasledovné stĺpce:

- *Month* – mesiac, v ktorom sa zájazd koná
- *Days* – počet dní zájazdu (vrátane cesty)
- *Capacity* – kapacita zájazdu
- *Customers* – počet zákazníkov, ktorý si zájazd zakúpili
- *Price* – cena zájazdu; na požiadanie zadávateľa zobrazená ako kategoriálny atribút
- *Transport* – typ dopravy
- *Locality* – lokalita, v ktorej sa zájazd koná
- *Country* – krajina, v ktorej sa zájazd koná
- *Food_Service* – stravovanie

6.3. Modelovanie

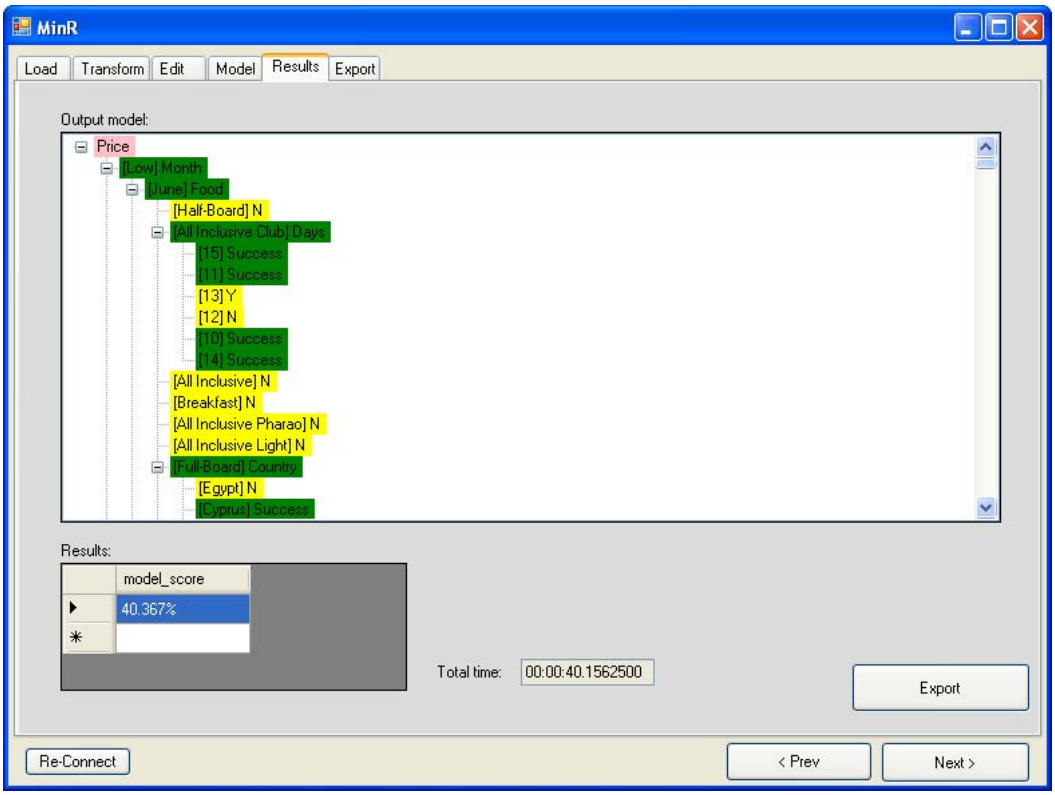
Klasifikačné algoritmy som podrobil testom na umelých dátach. Výsledky sú popísané v kapitole 4.6. Podľa výsledkov boli vybrané 3

modelovacie techniky: 2 algoritmy na generovanie rozhodovacích stromov (ID3, C4.5) a klasifikačná verzia algoritmu analýzy nákupného košíka (MBAC). Rozhodovacie stromy sú popísané v kapitole 4.1. Algoritmus MBAC som vytvoril v rámci výskumu a na dátach som sa pokúsil otestovať jeho použiteľnosť. Jeho detailný popis sa nachádza na strane 44.

Beh 1

Základný dataset neobsahuje cieľový atribút. Vytvoril som nový atribút *Success*, ktorý bude určovať úspešnosť zájazdu. Použitím SQL príkazu som v závislosti na pomere zákazníkov a kapacity zájazdu ako hodnotu nového atribútu doplnil úspech (Y) resp. neúspech (N). Dáta som exportoval do súboru „*data_analysis2_ds2.csv*“, ktorý sa nachádza v adresári „*data*“ na priloženom CD.

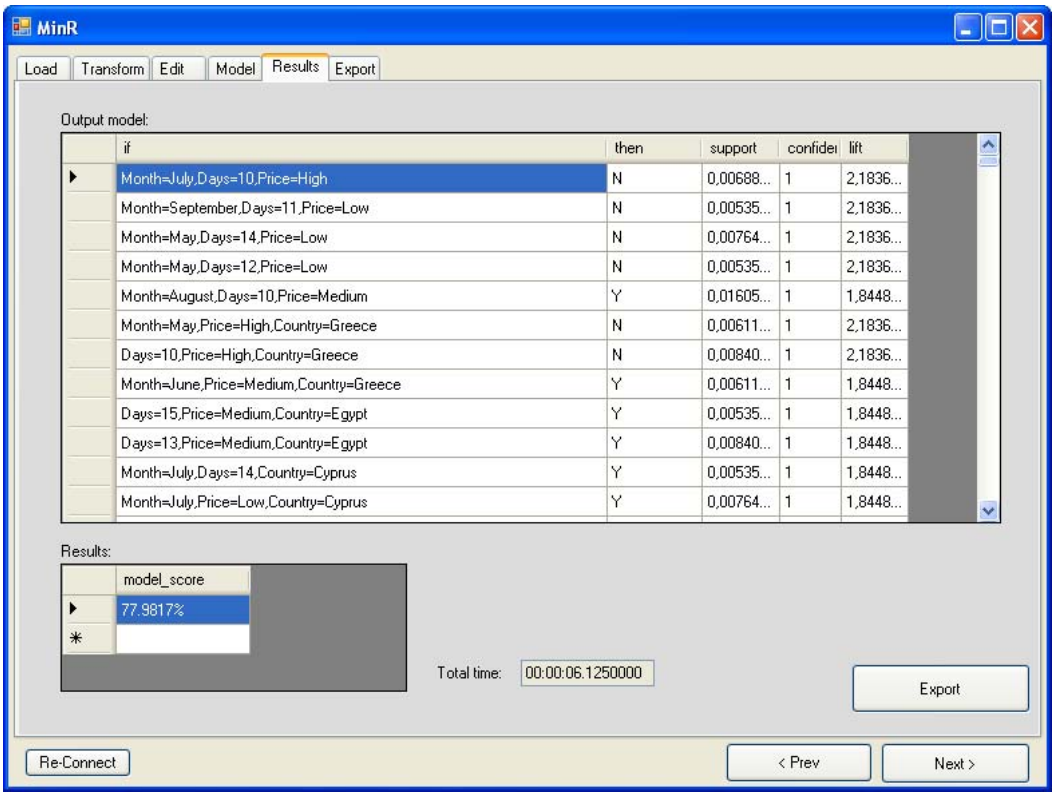
Takto upravené dáta som načítal do mojej aplikácie. Ako prvý som vyskúšal algoritmus na generovanie rozhodovacích stromov ID3.



obr. 22 – Výsledky modelovania pomocou ID3 algoritmu

Ako môžeme vidieť na obrázku, výsledky sú neuspokojivé. Úspešnosť tesne cez 40% je v prípade binárneho cieľového atribútu horšia ako náhodná voľba. V prípade algoritmu C4.5 boli výsledky o niečo lepšie. S celkovou správnosťou modelu 53.2% sa ale prirodzene nemožno uspokojiť.

Ako posledný som použil algoritmus MBAC. S úspešnosťou skoro 78% je tento model oveľa lepší ako predchádzajúce modely. Vidieť tiež, že čas modelovania bol viac ako 6x rýchlejší.



obr. 23 – Výsledky modelovania pomocou algoritmu MBAC

Predpokladám, že dôvodom neúspechu rozhodovacích stromov môžu byť veľké domény atribútov, ktoré vedú k veľkej košatosti stromov. Je preto potrebné sa vrátiť do fázy predspracovania dát a upraviť dáta.

Beh 2

Vytvoril som nový dataset, ktorý vznikol z predošlého odstránením atribútu lokality a zaradením krajín do 3 kategórii:

- *Common*
- *Average*
- *Premium*

Krajiny som do týchto kategórii rozdelil za pomoci zadávateľa, ktorý mal dobrý prehľad o tom, ako veľmi je ktorá krajina lukratívna. Dáta sa nachádzajú v súbore „*data_analysis2_ds3.csv*“, ktorý je uložený v adresári „*data*“ na priloženom CD.

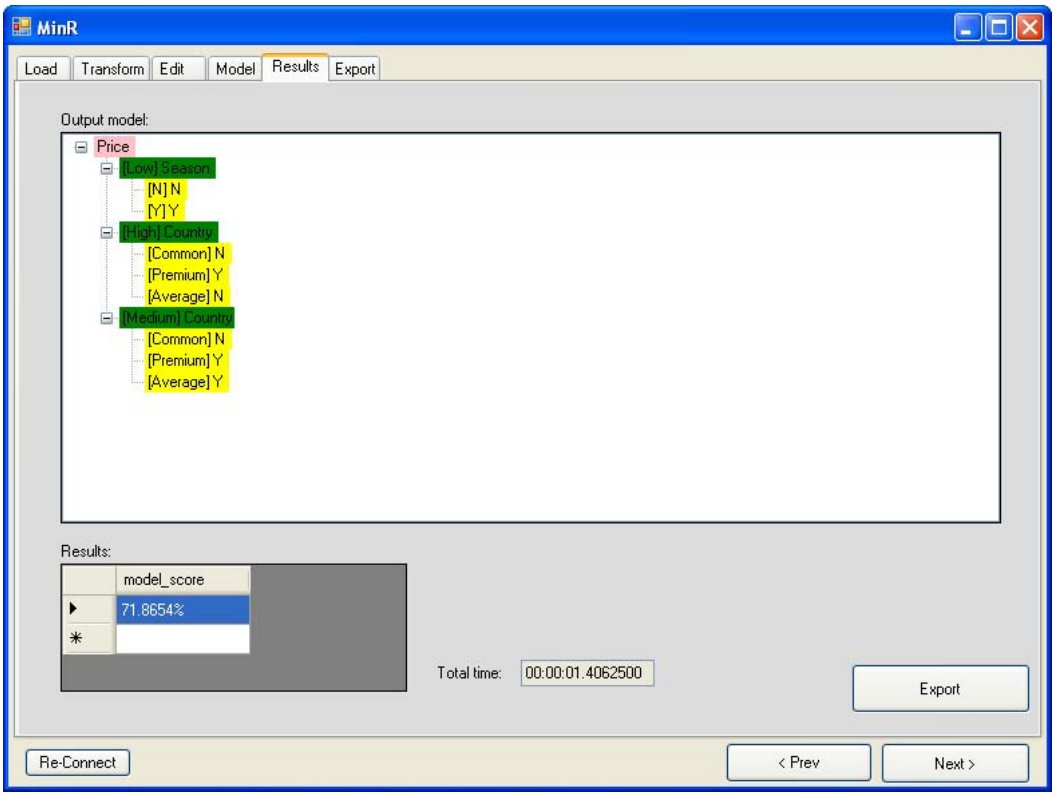
Výsledkom bolo zvýšenie celkovej správnosti rozhodovacieho stromu generovaného algoritmom ID3 na 48,318%. Správnosť modelov pre algoritmy C4.5 a MBAC zostala identická.

Beh 3

Stromy boli stále príliš košaté. Rozhodol som sa preto atribút *Month* transformovať na binárny atribút *Season*, ktorý bude určovať, či je zájazd v špičke sezóny (júl, august), alebo mimo nej (jún, september, ...). Dáta sa

nachádzajú v súbore „*data_analysis2_ds4.csv*“, ktorý je uložený v adresári „*data*“ na priloženom CD.

Model vytvorený algoritmom ID3 bol zhodný s predošlým behom. Pre algoritmus C4.5 som skúsil nastaviť maximálnu hĺbku stromu najskôr na 3. To viedlo k výrazne lepšej spoľahlivosti. Výsledok modelovania vidíme na obr. 24.



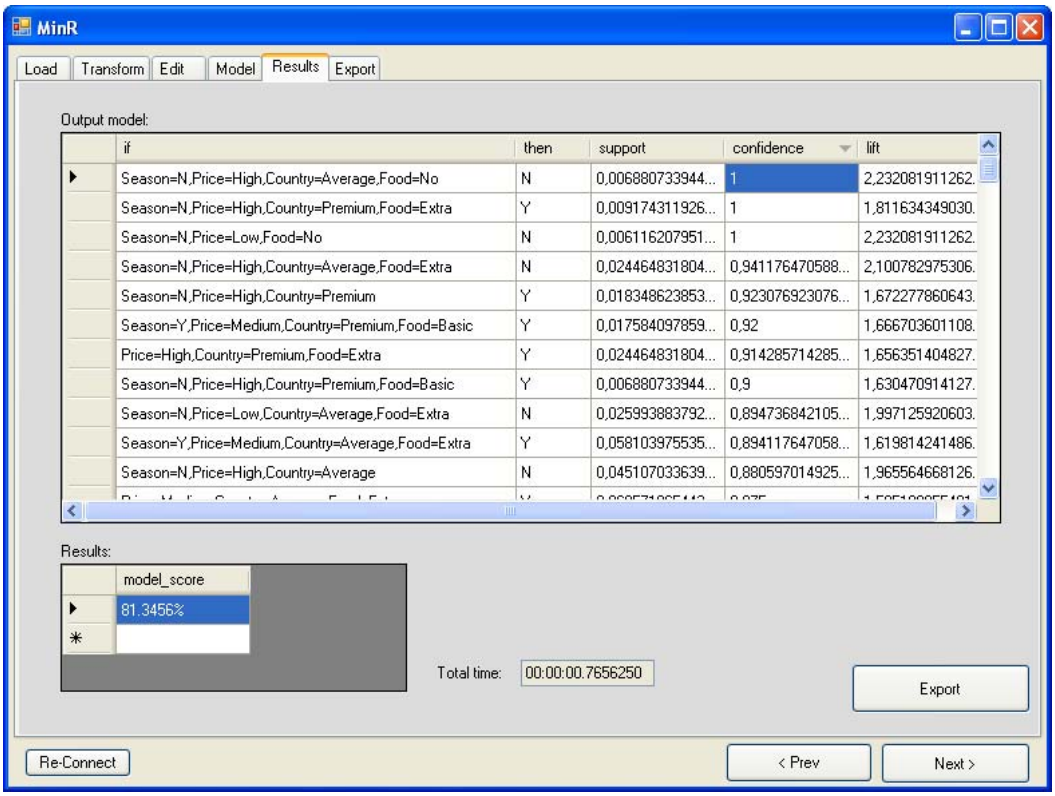
obr. 24 - Výsledok modelovania C4.5 s maximálnou hĺbkou stromu 3

Po úspechu som sa rozhodol otestovať aj maximálnu hĺbku 2. Výsledkom bol model s celkovou správnosťou tesne pod 75%. Rozhodovací strom určoval správnosť zájazdu na základe jediného atribútu – ceny. Napriek celkom vysokej celkovej správnosti modelu je takýto model slabo použiteľný. To sa potvrdilo aj po konzultácii zo zadávateľom, pre ktorého nebola táto informácia ničím novým.

Beh 4

Na záver som sa rozhodol z predošlého datasetu odstrániť atribút *Days*, ktorý určoval dĺžku zájazdu. Atribút *Food* som transformoval na atribút obsahujúci 3 kategórie: *No* (žiadne stravovanie), *Extra* (variácie All Inclusive stravovania) a *Basic* (polpenzia, plná penzia, raňajky). Tento dataset sa nachádza v súbore „*data_analysis2_ds5.csv*“, ktorý je uložený v adresári „*data*“ na priloženom CD.

Pre algoritmy ID3 a C4.5 bol prínos zanedbateľný. Algoritmus MBAC však dosiahol veľmi dobré výsledky.



obr. 25 - Výsledky algoritmu MBAC na poslednom datasete

Celková správnosť pre model presiahla 81%. Zaujímavé sú hlavne prvé 3 pravidlá, ktoré dosiahli 100% spoľahlivosť. Detailnejším preskúmaním najlepších pravidiel spolu so zadávateľom sa mi poradilo extrahovať nasledujúce informácie:

- Ak má byť zájazd mimo sezónu úspešný, musí byť do exkluzívnej krajiny a zároveň mať vysokú cenu
- V sezóne sa najlepšie predávajú zájazdy s nízkou cenou

Druhá informácia patrí medzi očakávané a nebola pre zadávateľa až tak prekvapujúca. Na druhú stranu prvú znalosť považoval zadávateľ za prínos a užitočnú informáciu, ktorú zohľadní v ponukách mimosezónnych zájazdov.

Záver

Algoritmy na generovanie rozhodovacích stromov dávali na umelých dátach vynikajúce výsledky. Na reálnych dátach však svoje kvality nepotvrdili. Prekvapila hlavne úspešnosť vlastného algoritmu MBAC, ktorý dosiahol výborné výsledky. Jeho nevýhodou je ale veľkosť modelu a slabá vizuálna reprezentácia modelu.

7. Vytvorená aplikácia

7.1. Koncept

Úvod

Súčasťou mojej diplomovej práce je aplikácia, ktorá prezentuje nadobudnuté znalosti. Aplikácia implementuje viacero algoritmov a techník na predspracovanie dát, modelovanie, testovanie a porovnávanie jednotlivých algoritmov.

Samotná aplikácia bola vytvorená vo vývojovom prostredí *Microsoft Visual Studio C# 2008 Express Edition*. Dôvodom pre túto voľbu bola hlavne voľná dostupnosť a kompaktnosť tohto vývojového nástroja.

Aplikácia používa *Microsoft SQL Server 2005 Express* ako pracovnú databázu. V databáze sú uložené niektoré aplikačné nastavenia a veľké množstvo uložených databázových procedúr, ktorých význam je objasnený v nasledujúcich odsekoch.

Aplikácia má teda dve vrstvy – aplikačnú a databázovú. Aplikačná vrstva, vytvorená v jazyku C#, slúži hlavne ako užívateľské rozhranie, ktoré umožňuje jednoducho a prehľadne načítavať dáta, vykonávať transformácie a predspracovanie dát, spúšťať modelovanie použitím zvolených algoritmov a zobrazovať výsledky. Okrem implementácie algoritmu k-najbližších susedov sú všetky algoritmy implementované v podobe databázových procedúr, ktoré sú volané aplikáciou. Algoritmus k-najbližších susedov je implementovaný aplikačne aj ako databázová procedúra. To mi poslúžilo ako test porovnania rýchlosti týchto algoritmov.

Databázové procedúry

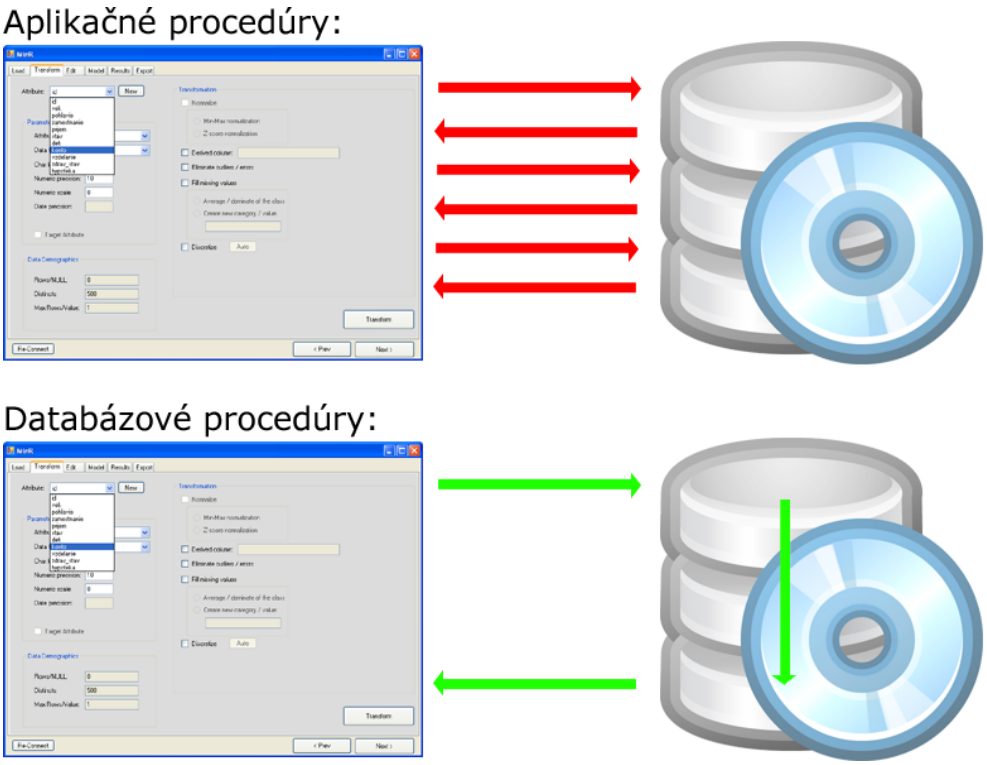
Dôvodov pre rozhodnutie naprogramovať algoritmy na predspracovanie dát a modelovanie vo forme uložených databázových procedúr je niekoľko. Hlavným dôvodom je fakt, že väčšina týchto algoritmov používa operácie nad celou množinou dát (masové operácie). Keďže pri dobývaní znalostí bývajú tieto dáta rozsiahle a databázové systémy sú na tento typ operácií optimalizované, prišlo mi toto rozhodnutie logické. Predpokladal som, že takáto implementácia bude rýchlejšia ako jej aplikačný variant.

Ďalším dôvodom je šetrenie komunikačného kanálu. Ako zdroj dát pri dobývaní znalostí je často použitá existujúca databáza, najčastejšie dátový sklad[12]. K tomuto dátovému skladu sa pristupuje vzdialene[13]. Ak je algoritmus implementovaný na strane aplikačného serveru, musia dáta často putovať

[12] Dátový sklad (Data Warehouse) je relačný databázový systém, ktorý umožňuje riešiť úlohy analytického charakteru nad rozsiahlymi súbormi dát.

[13] Aplikácia na dobývanie znalostí sa zvyčajne spúšťa z jedného servera (aplikačný) a dátový sklad beží na druhom (databázový). K dátam sa preto pristupuje cez komunikačný kanál napr. lokálnu sieť.

medzi aplikačným a databázovým serverom. V prípade použitia databázovej procedúry aplikácia zavolá túto procedúru, všetky výpočty a operácie prebehnú na databázovom serveri a následne sú výsledky vrátené aplikačnému serveru. Rozdiel medzi použitím aplikačných a databázových procedúr je zobrazený na obr. 26.



obr. 26 – Rozdiel medzi použitím aplikačných a databázových procedúr

V neposlednej rade tento prístup zjednodušuje transakčné spracovanie dát.

7.2. Prihlásenie do aplikácie

Po spustení aplikácie sa objaví prihlasovacie okno. Okno obsahuje textové polia pre zadanie prihlasovacieho mena, hesla, názvu servera a názvu pracovnej databázy. Aplikácia sa pokúsi načítať názov serveru a názov databázy zo súboru, kde sú automaticky uložené posledné funkčné hodnoty. V prípade úspechu sa spodná časť okna schová. Textové polia pre zadanie názvu servera a pracovnej databázy možno skrývať/zobrazovať pomocou tlačidiel *More* a *Less*.

V ľavej dolnej časti okna sa nachádza tlačidlo *Connect*, ktoré slúži na odoslanie zadaných údajov. Súčasne sa aplikácia pokúsi o pripojenie k pracovnej databáze. V prípade neúspechu je užívateľovi zobrazená chybová správa. Kým nie je funkčné spojenie s pracovnou databázou, aplikácia neumožní žiadne akcie. Jediný funkčný ovládací prvok hlavného okna bude tlačidlo *Re-Connect*.

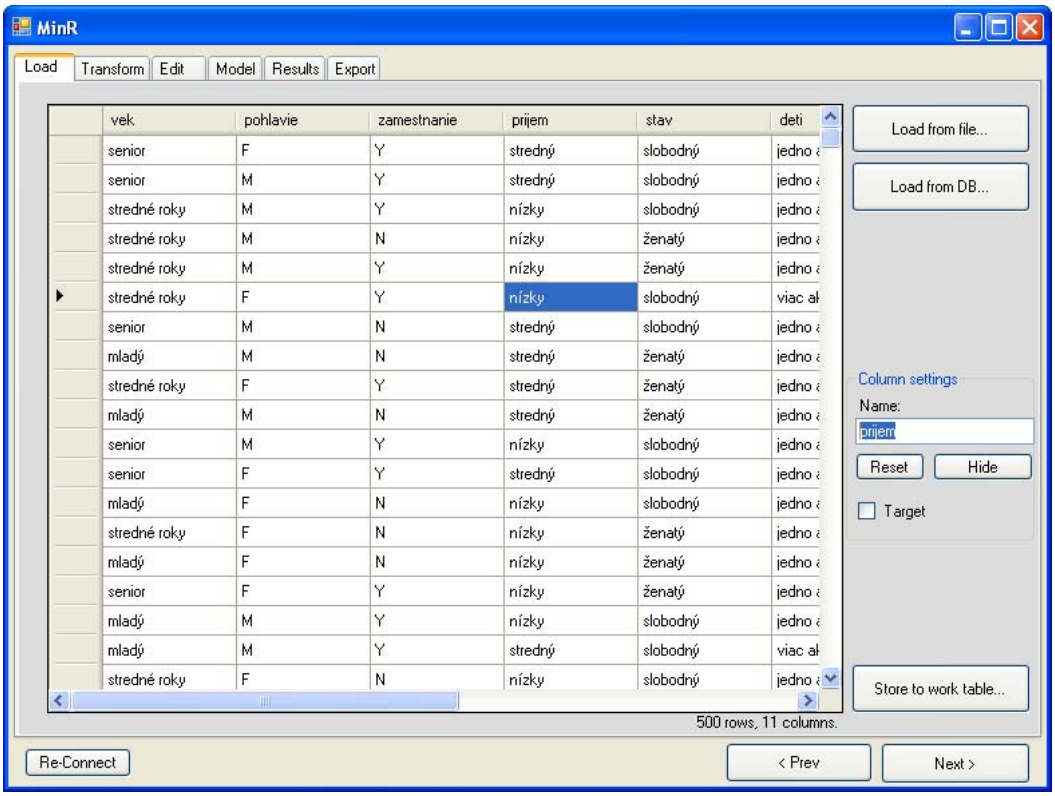
obr. 27 – Prihlasovacie okno

Ak sa pripojenie podarí, užívateľovi sa zobrazí funkčné hlavné okno aplikácie. Okno obsahuje 6 záložiek: *Load*, *Transform*, *Edit*, *Model*, *Results* a *Export*. V ľavom dolnom rohu je tlačidlo pre zmenu pripojenia k pracovnej databáze. V pravom dolnom rohu sú tlačidlá *Prev* a *Next*, ktoré umožňujú pohyb medzi záložkami.

7.3. Načítanie dát

Záložka *Load* umožňuje užívateľovi načítať dáta. Podporované sú dva zdroje dát: databázový systém alebo textový súbor. Pre načítanie dát z databázového systému slúži tlačidlo „*Load from DB*“, pre súbor je to „*Load from file*“. V oboch prípadoch sa zobrazí príslušné okno, ktoré umožní užívateľovi vyplniť potrebné informácie na načítanie dát. Funkčnosť týchto okien je popísaná v nasledujúcich podkapitolách.

Po úspešnom načítaní sa užívateľovi dáta zobrazia v dátovej mriežke a sprístupní sa skupina ovládacích prvkov na editáciu stĺpcov. V textovom poli *Name* môže užívateľ vidieť a meniť názov aktuálne označeného stĺpca. V prípade nechcenej editácie názvu je možné túto editáciu odvolať tlačidlom *ESC*.

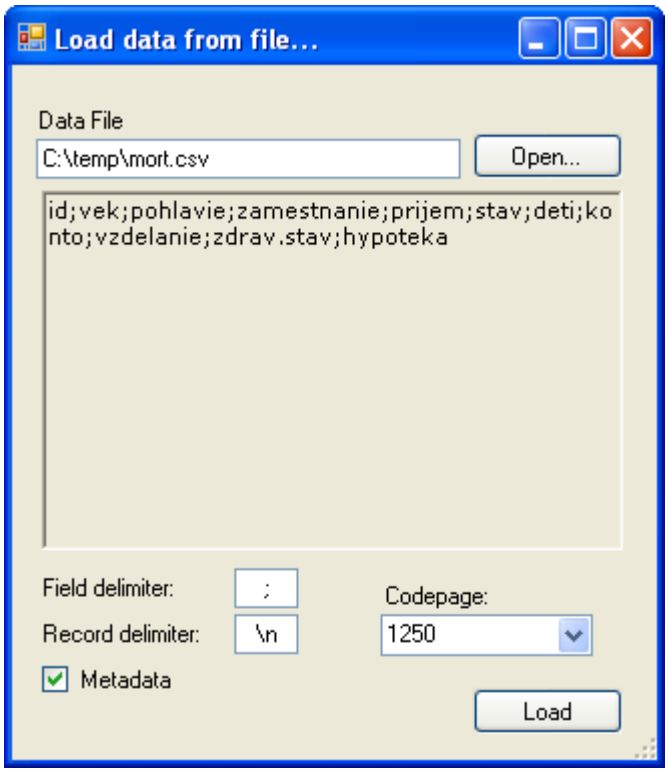


obr. 28 – Záložka pre načítanie dát

Tlačidlo *Hide* umožňuje užívateľovi skryť stĺpce, ktoré obsahujú dáta nepotrebné pre modelovanie. Ak užívateľ omylom skryje potrebný stĺpec, môže odkryť späť všetky stĺpce tlačidlom *Reset*. Zaškrtávacie pole *Target* slúži na označenie cieľového atribútu v prípade klasifikácie.

Načítanie dát zo súboru

Okno pre načítanie dát zo súboru používa komponent *OpenFileDialog*, ktorý zobrazí dialógové okno a umožní užívateľovi zvoliť súbor obsahujúci dáta. Dialógové okno sa aktivuje tlačidlom *Open*. Následne sa v textovom poli zobrazí prvý riadok súboru. Užívateľ zvolí separátor stĺpcov (*Field delimiter*) a separátor záznamov (*Record delimiter*). Ak prvý riadok obsahuje názvy stĺpcov, treba označiť zaškrtávacie pole *Metadata*. Na záver treba zvoliť správne kódovanie *Codepage*.



obr. 29 – Načítanie dát zo súboru

Stlačením tlačidla *Load* sa zavolá metóda *openFile* hlavného formulára, ktorá najskôr vytvorí súbor *schema.ini*, ktorý obsahuje všetky potrebné informácie. Potom použije triedu *OleDbConnection* a načíta dáta do dátovej mriežky hlavného okna aplikácie. Súčasne si uloží do poľa reťazcov pôvodné názvy stĺpcov. Toto pole sa neskôr použije pri generovaní textového reťazca na premenovanie stĺpcov. V prípade úspechu týchto operácii sa okno zavrie, v opačnom prípade je zobrazená chybová správa s popisom chyby, ktorá nastala.

Po stlačení tlačidla „*Store to work table*“, ktoré sa nachádza v pravom dolnom rohu hlavnej obrazovky, sa spustí databázová procedúra ***sp_createWorkFromFile***. Táto procedúra má ako parametre cestu k súboru, názov súboru, textový reťazec na vytvorenie tabuľky a premenovanie stĺpcov, oddeľovač stĺpcov, oddeľovač záznamov, kódovanie a príznak, či prvý riadok obsahuje názvy stĺpcov alebo nie.

Uložená procedúra načíta riadky súboru do tabuľky *wrkTable_temp* pomocou SQL príkazu *BULK INSERT*. Následne sú jednotlivé stĺpce na základe zvoleného oddeľovača uložené do pracovnej tabuľky *wrkTable*.

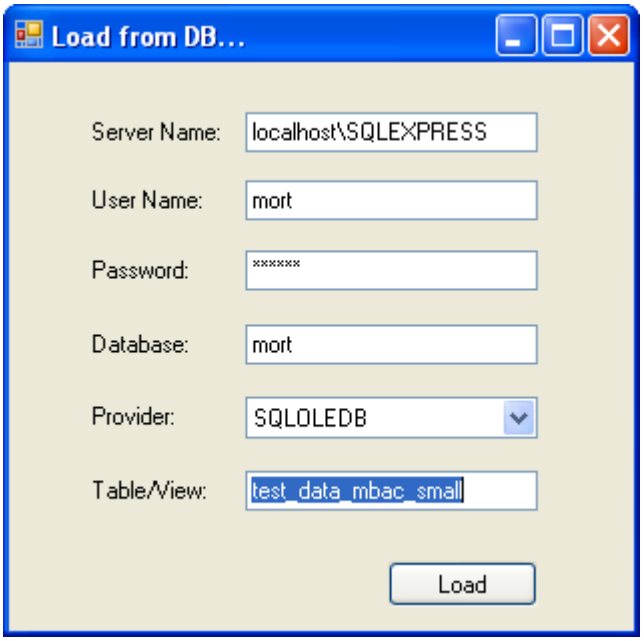
Načítanie dát z databázy

Na obr. 30 vidieť aplikačné okno, ktoré slúži na načítanie dát z databázy. Podporované databázové systémy sú *Oracle*, *MS SQL Server* a *Teradata*. Okno obsahuje celkovo 6 polí:

- *Server Name* – meno databázového servera

- *User Name* – užívateľské meno
- *Password* – heslo
- *Database* – názov databázy
- *Provider* – zo zoznamu treba vybrať správneho „providera“ pre pripojenie
- *Table/View* – názov tabuľky alebo pohľadu, ktorý obsahuje požadované dáta

Funkčné pripojenie sa uloží to tabuľky *wrkLastCon* v pracovnej databáze. Pri každom otvorení okna sú v daných poliach vždy predvyplnené posledné funkčné hodnoty.



obr. 30 – Načítanie dát z databázy

V pravom dolnom rohu obrazovky sa nachádza tlačidlo *Load*. Po jeho stlačení sa zavolá metóda *openDB* hlavného formulára s parametrami zadanými v tomto okne. Táto metóda spustí uloženú databázovú procedúru ***sp_defineSource***, ktorá použije systémové databázové procedúry ***sp_addlinkedserver*** a ***sp_addlinkedsrvlogin*** na vytvorenie pripojenia k vzdialenému serveru. Následne načíta dáta do dátovej mriežky hlavného okna aplikácie. Súčasne si uloží do poľa reťazcov pôvodné názvy stĺpcov. Toto pole sa neskôr použije pri generovaní textového reťazca na premenovanie stĺpcov. V prípade úspechu týchto operácii sa okno zavrie, v opačnom prípade je zobrazená chybová správa s popisom chyby, ktorá nastala.

Po stlačení tlačidla „*Store to work table*“, ktoré sa nachádza v pravom dolnom rohu hlavnej obrazovky, sa spustí databázová procedúra ***sp_createWorkFromTable***. Táto procedúra má ako parametre názov tabuľky, textový reťazec na premenovanie stĺpcov a názov pripojenia

vytvoreného procedúrou ***sp_defineSource***. Uložená procedúra načíta dáta z vybranej tabuľky, premenuje stĺpce a uloží dáta do pracovnej tabuľky *wrkTable*.

Vytvorenie atribútov

Po uložení dát do pracovnej tabuľky *wrkTable* je zavolaná databázová procedúra ***sp_preprocess***. Táto procedúra ma 2 parametre: názov tabuľky (*wrkTable*) a názov stĺpca, ktorý odpovedá cieľovému atribútu. V prípade, že cieľový atribút nebol označený, zvolí sa ako hodnota tohto parametru prázdny textový reťazec.

Procedúra najskôr pridá do tabuľky *wrkTable* automaticky generovaný primárny kľuč. Následne pre každý atribút uloží informácie o dátovom type a spočíta demografické dáta: počet záznamov s *NULL* hodnotou, počet unikátnych hodnôt a maximálny počet záznamov s rovnakou hodnotou. Potom sa pokúsi automaticky identifikovať typ atribútu. Atribút môže byť kategoriálny alebo spojitý. Všetky tieto údaje sú uložené v tabuľke *wrkAttr*. Táto tabuľka je cudzím kľúčom spojená s tabuľkami *wrkAttrDtype* a *wrkAttrType*. Tabuľka *wrkAttrDtype* je číselník podporovaných dátových typov a *wrkAttrType* číselník typov atribútov.

7.4. Predspracovanie dát

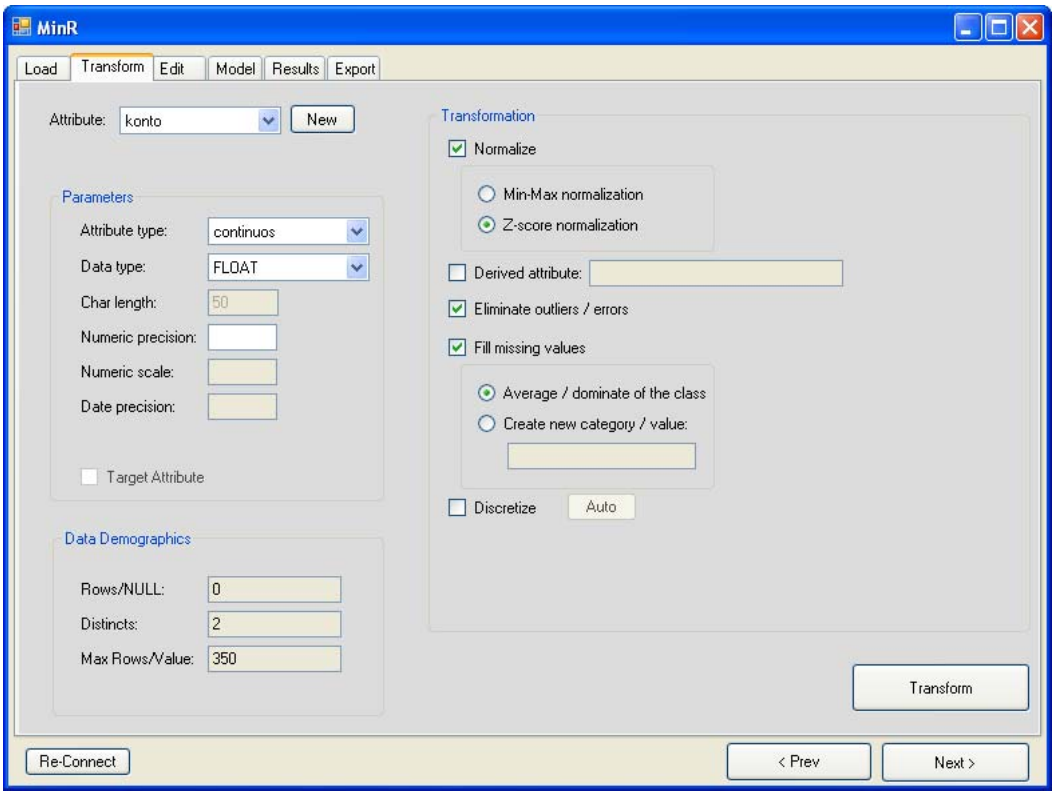
Záložka *Transform* obsahuje ovládacie prvky, ktoré umožňujú upraviť rôzne charakteristiky jednotlivých atribútov a nadefinovať potrebné transformácie. V ľavom hornom rohu záložky sa nachádza komponent *ComboBox*, ktorý obsahuje zoznam atribútov. Napravo od neho je tlačidlo *New*, ktoré umožňuje pridať nový atribút. Pri pridávaní sa zobrazí textové pole, kde je potrebné zadať názov nového atribútu a potvrdiť tlačidlom *Ok*.

Nižšie sa nachádza skupina ovládacích prvkov, ktoré umožňujú meniť typ atribútu, dátový typ atribútu a charakteristiky dátového typu, ako dĺžka reťazca alebo numerická presnosť desatinného čísla. V ľavom dolnom rohu sa nachádzajú 3 textové polia, ktoré sú určené iba na čítanie. Zobrazujú demografické štatistiky pre daný atribút. Napravo sa nachádza skupina ovládacích prvkov *Transformation*. Tu užívateľ definuje transformácie, ktoré si želá na danom atribúte vykonať.

Všetky nastavenia sa ukladajú do tabuľky *wrkAttr* a po stlačení tlačidla *Transform*, ktoré sa nachádza v pravom dolnom rohu záložky, dôjde k vykonaniu zvolených transformácií zavolaním databázovej procedúry ***sp_transform***. Užívateľ môže vykonať nasledujúce transformácie:

- Normalizácia (*Normalize*)
- Vytvorenie odvodeného atribútu (*Derive attribute*)
- Eliminácia chýb a odľahlých hodnôt (*Eliminate outliers*)
- Doplnenie chýbajúcich hodnôt (*Fill missing values*)
- Diskretizácia (*Discretize*)

Pri každej transformácii sa nachádza zaškrtávacie pole. Ak má byť transformácia pre zvolený atribút vykonaná, treba ho zaškrtnúť.



obr. 31 – Záložka na predspracovanie dát

Normalizácia

Normalizácia je prístupná iba pre atribúty, ktoré majú dátový typ *NUMERIC* alebo *FLOAT*. Užívateľ musí zvoliť, akú normalizáciu chce vykonať. Implementované sú 2 varianty: min-max normalizácia a normalizácia podľa smerodajnej odchýlky (*zScore*). V prvom prípade sa najskôr získa pomocou databázových funkcií maximálna (*MAX*) a minimálna (*MIN*) hodnota, v druhom priemer (*AVG*) a rozptyl (*STDEV*). Potom sa vykoná úprava hodnôt použitím príslušného vzorca.

Odvodený atribút

Do textového poľa treba zadať platný SQL reťazec na vytvorenie odvodeného atribútu. Ak by napríklad užívateľ mal atribút *RC* obsahujúci rodné čísla a chcel atribút obsahujúci mesiac narodenia, môže vytvoriť nový atribút *MESIAC_NAR* a použiť reťazec „SUBSTRING(RC,3,2)“.

Eliminácia chybných/odľahlých hodnôt

Ak je príslušné zaškrtávacie pole označené, vykoná aplikácia pre zvolený atribút automatickú elimináciu. Ak je atribút kategoriálny, odstráni aplikácia chybné hodnoty. Za chybnú kategoriálnu hodnotu sa považuje taká hodnota, ktorej frekvencia je menšia ako 1%.

Ak je zvolený atribút spojitý, vykoná sa eliminácia odľahlých hodnôt. Za odľahlú hodnotu sa v tomto prípade považuje hodnota, ktorá je od strednej hodnoty pre tento atribút vzdialená o viac ako 5*rozptyl pre tento atribút.

Eliminované hodnoty budú nahradené hodnotou *NULL*. Preto aplikácia vyžaduje v prípade použitia automatickej eliminácie aj použitie automatického doplňovania chýbajúcich hodnôt.

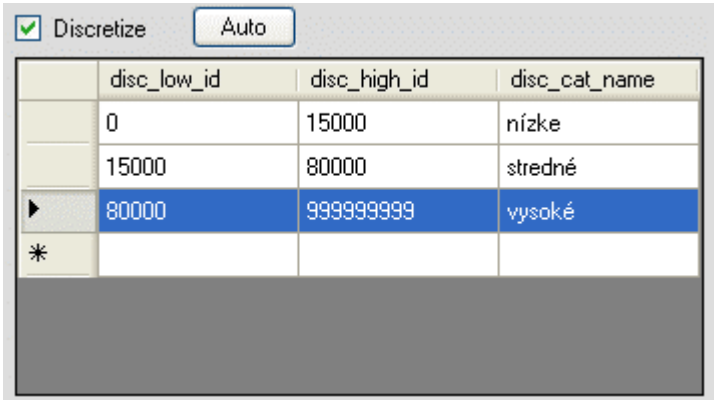
Doplňovanie chýbajúcich hodnôt

Operácie, ktoré aplikácia pri dopĺňaní chýbajúcich hodnôt vykoná, sa líšia v závislosti od toho, či je atribút kategoriálny alebo spojitý. Pre kategoriálny atribút sa užívateľ musí rozhodnúť, či chce pre chýbajúce hodnoty vytvoriť novú kategóriu, alebo chce doplniť najfrekventovanejšiu kategóriu. V oboch prípadoch procedúra *sp_preprocess* zavolá dynamicky generovaný SQL reťazec, ktorým doplní želanú hodnotu.

Podobne tomu je u spojitých dát, kde sa užívateľ môže rozhodnúť medzi doplnením zvolenej konštantnej hodnoty alebo doplnením priemernej hodnoty.

Diskretizácia

Diskretizácia je prístupná iba pre spojité numerické atribúty. V dolnej časti tejto záložky sa nachádza dátová mriežka, ktorá je prepojená s tabuľkou *wrkAttrDisc*. Táto slúži na definíciu intervalov a názvov pre vytvárané kategórie.



obr. 32 – Diskretizácia

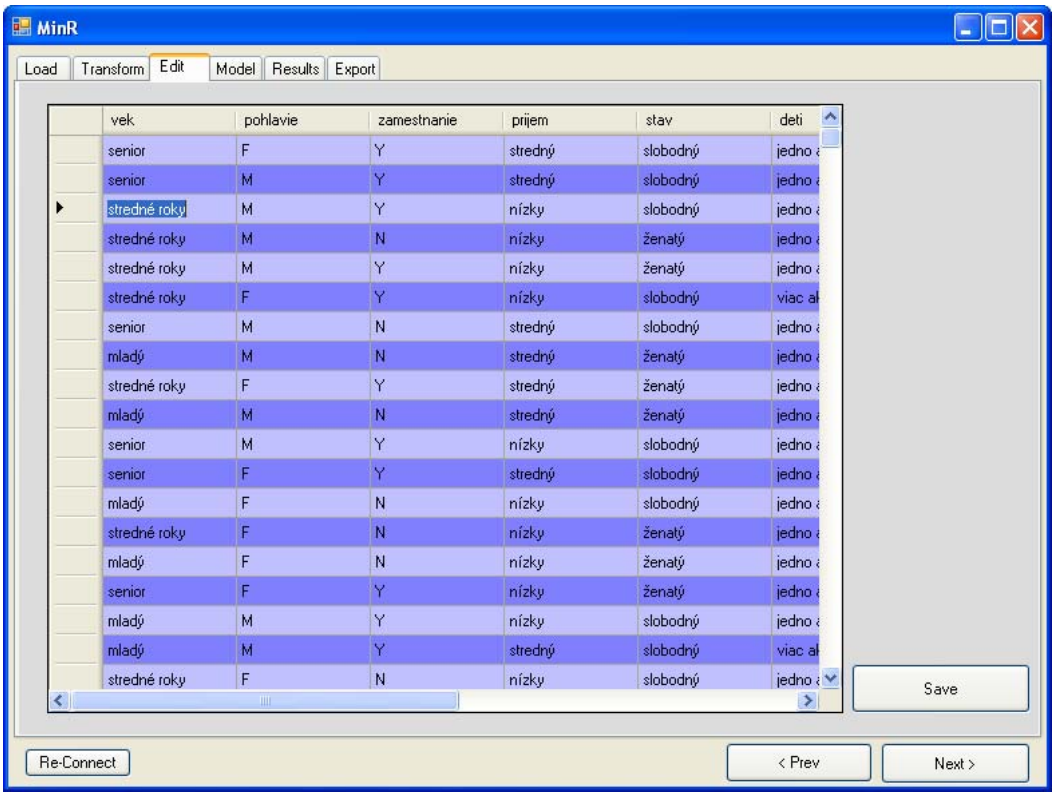
Intervaly možno automaticky identifikovať použitím algoritmu KEX, ktorý je popísaný na strane 19. Podmienkou je ale existencia kategoriálneho cieľového atribútu. Algoritmus je implementovaný v podobe databázovej procedúry s názvom ***sp_transKEX***. Ako parametre požaduje názov tabuľky, názov cieľového atribútu a ID atribútu, pre ktorý sa hľadajú kategórie.

Táto databázová procedúra používa 3 pomocné tabuľky *trans_kex_intervals*, *trans_kex_intervalsMerge* a *trans_kex_intervalsTemp* a pomocnú procedúru ***sp_transKEXMerge***, ktorá slúži na spájanie susedných intervalov podľa podmienok definovaných algoritmom. Na záver sú

identifikované kategórie uložené do tabuľky *wrkAttrDisc*. Užívateľovi sa zobrazia v dátovej mriežke v pravom dolnom rohu a umožnia mu upraviť názvy a intervaly pre každú kategóriu.

7.5. Editácia dát

Záložka *Edit* obsahuje dátovú mriežku (obr. 33), ktorá po vykonaní všetkých transformácii zobrazí dáta a umožní ich editáciu. V pravom dolnom rohu je tlačidlo *Save*, ktoré uloží vykonané zmeny. V tejto fáze má užívateľ pripravené dáta na modelovanie. Zobrazí sa mu záložka *Model*.

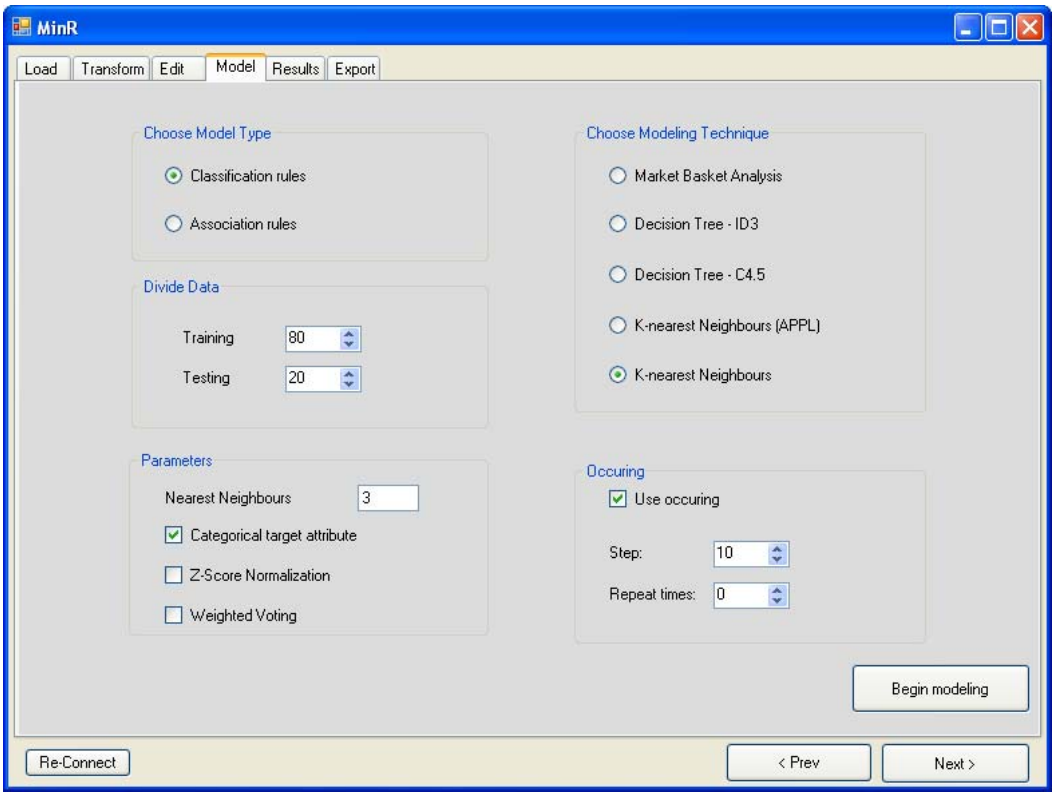


obr. 33 – Záložka na editáciu dát

7.6. Modelovanie a výsledky

Záložka na modelovanie obsahuje niekoľko skupín ovládacích prvkov. V ľavom hornom rohu sa nachádza prepínač typu modelovania. Podporované sú klasifikačné (*Classification rules*) a asociačné pravidlá (*Association rules*). Pravý horný roh obsahuje prepínač konkrétneho algoritmu. Na generovanie asociačných pravidiel je prístupný iba algoritmus MBA. Pri klasifikácii umožňuje aplikácia užívateľovi zvoliť: MBA pre klasifikáciu, algoritmus ID3, algoritmus C4.5, algoritmus k-najbližších susedov a jeho aplikačný variant (APPL), ktorý bol implementovaný za účelom porovnania rýchlosti.

Vľavo sa v prípade voľby klasifikačných pravidiel zobrazia 2 polia, ktoré umožňujú užívateľovi aplikácie voľbu veľkosti trénovacej a testovacej množiny v percentách. Prednastavené hodnoty sú 80 / 20.



obr. 34 – Záložka na nastavenia modelovacej techniky

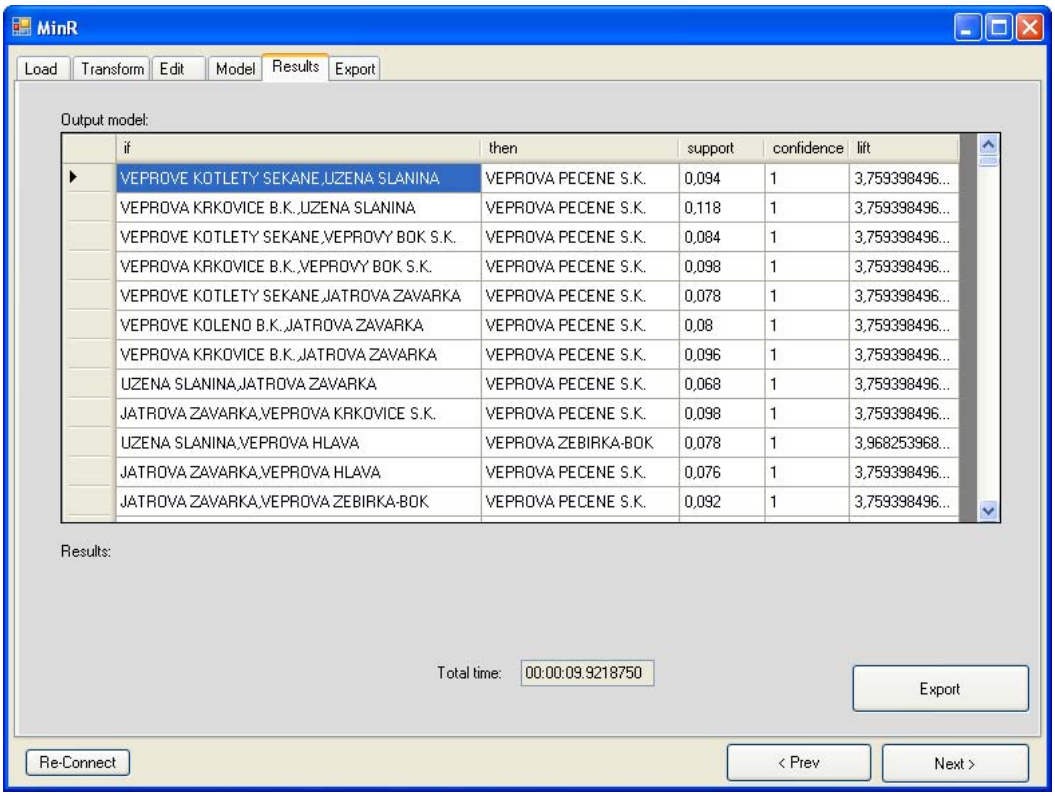
V pravom dolnom rohu sa nachádza skupina ovládacích prvkov s názvom *Occuring*. Je prístupná iba v prípade, že zvolený typ modelu sú klasifikačné pravidlá. Slúži na spustenie viacerých behov toho istého algoritmu nad rovnakými dátami s rôznymi parametrami. To slúži na testovanie algoritmu. Užívateľ musí nastaviť parameter *Step*, ktorý určuje, po akých veľkých krokoch (v percentách) sa bude veľkosť trénovacej množiny zmenšovať medzi jednotlivými behmi. Parameter *Repeat times* určuje, koľkokrát bude algoritmus spustený v jednom behu.

V ľavom dolnom rohu sa zobrazuje pre každý algoritmus špecifická skupina ovládacích prvkov, ktorá slúži na nastavenie zvyšných parametrov algoritmu.

Proces modelovania sa spúšťa tlačidlom „*Begin modeling*“. Po úspešnom zbehnutí príslušného algoritmu sa zobrazí záložka *Results*. Táto záložka obsahuje výsledky modelovacieho procesu a je špecifická pre každý algoritmus.

Analýza nákupného košíka (MBA)

Algoritmus je implementovaný ako databázová procedúra. Detaily implementácie sú popísané na strane 41. Parametrami algoritmu sú minimálna podpora pre triviálne pravidlá, minimálna podpora pre komplexné pravidlá, minimálna spoľahlivosť a minimálne zlepšenie.

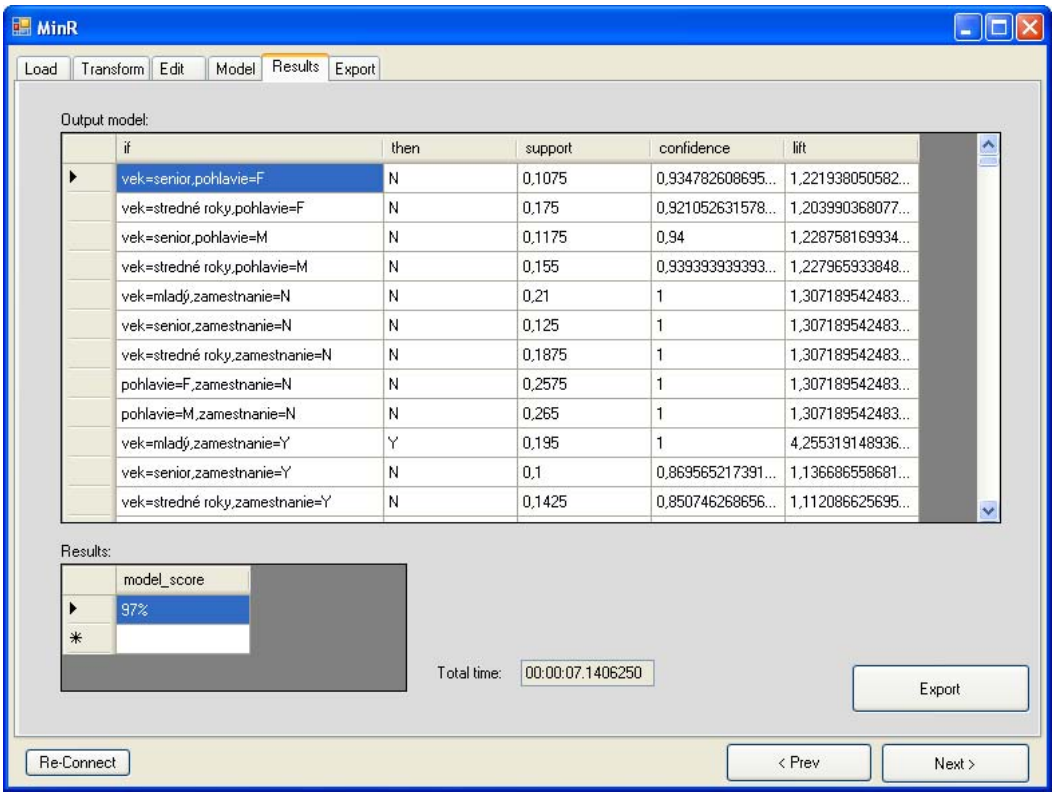


obr. 35 – Výsledky modelovania pomocou algoritmu MBA

Na záložke pre výsledky sa po úspešnom zbehnutí algoritmu objavia nadobudnuté pravidlá spolu s kritériami hodnotenia pravidiel – podpora, spoľahlivosť a zlepšenie. V spodnej časti nájdeme celkový čas.

MBA klasifikácia

Pri implementovaní analýzy nákupného košíka ma napadlo, že by bolo možné použiť túto metódu aj na klasifikáciu. Túto myšlienku som rozvinul a popísal na strane 44. Parametre algoritmu sú zhodné s klasickou verziou algoritmu MBA. Jediným rozdielom je prítomnosť testovacej fázy, takže je potrebné zvoliť percentuálnu deľbu dát na testovacie a trénovacie.



obr. 36 – Zobrazenie pravidiel pre klasifikáciu

Po spustení a prebehnutí algoritmu sa zobrazia klasifikačné pravidlá. V ľavom dolnom rohu sa nachádza informácia o celkovej správnosti (skóre) modelu.

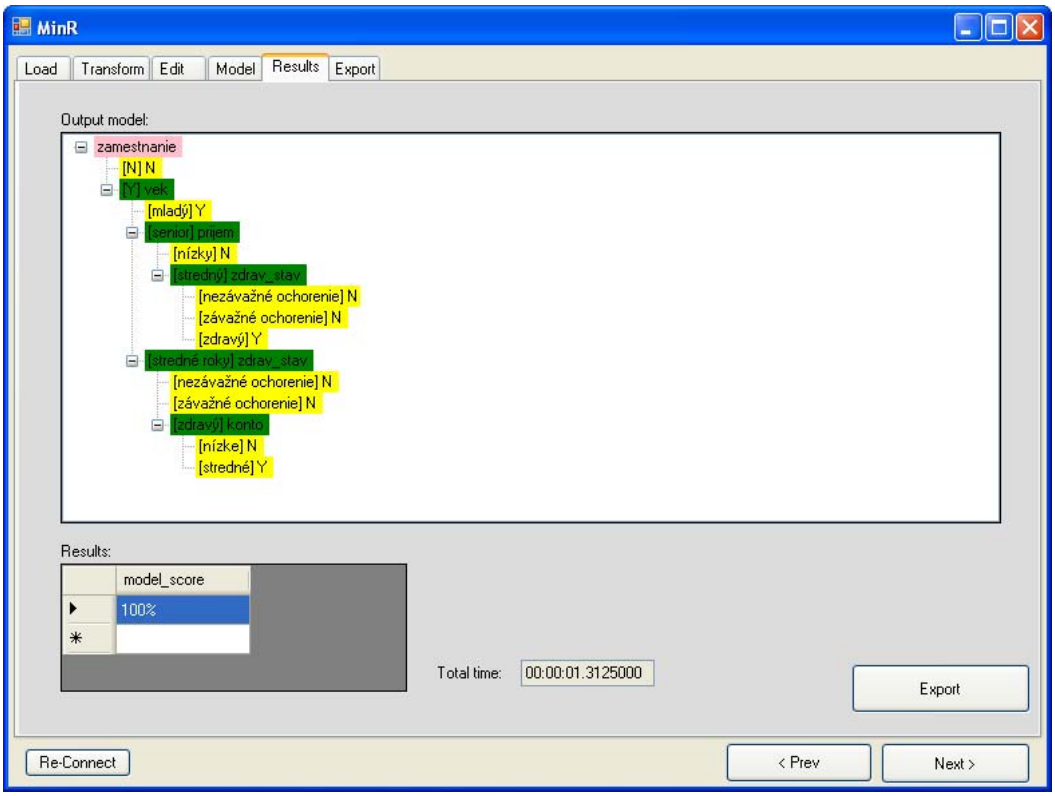
Algoritmus ID3

ID3 je algoritmus na tvorbu rozhodovacích stromov. Detailne bol popísaný v kapitole o rozhodovacích stromoch na strane 28. Algoritmus je implementovaný vo forme databázovej procedúry ***sp_id3Main***. Procedúra má 5 parametrov:

- Názov tabuľky, ktorá obsahuje dáta
- Názov cieľového atribútu
- Názov ID atribútu – to je taký stĺpec v tabuľke s dátami, ktorý nie je chápaný ako vstupný atribút, ale je použitý na jednoznačnú identifikáciu záznamu
- Veľkosť trénovacej množiny v %
- Veľkosť testovacej množiny v %

Príklad volania procedúry vyzerá nasledovne:

```
execute sp_id3Main 'data_sample_classif','hypoteka','id',100,0
```



obr. 37 – Výsledný rozhodovací strom pre algoritmus ID3

Na záložke *Results* sa po prebehnutí algoritmu objaví farebný rozhodovací strom. Koreňový uzol je ružovej farby, vnútorné uzly stromu sú zelenej farby a listy žltej farby. Na vykreslenie stromu bol použitý komponent *TreeView*.

Algoritmus C4.5

Algoritmus C4.5 je ďalším algoritmom na tvorbu rozhodovacieho stromu. Na rozdiel od algoritmu ID3 pripúšťa aj spojité vstupné atribúty. Detailný popis algoritmu sa nachádza v kapitole o rozhodovacích stromoch na strane 29.

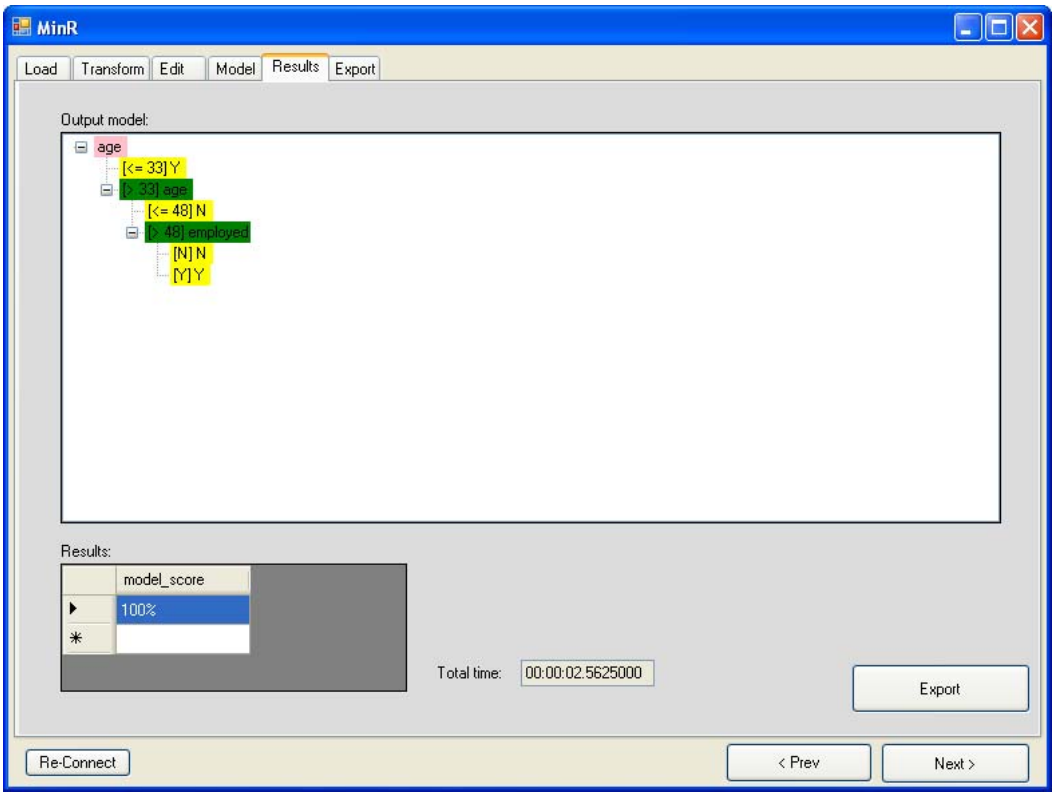
Algoritmus sa spúšťa zavolaním databázovej procedúry ***sp_c45Main***. Táto procedúra má nasledovné parametre:

- Názov tabuľky
- Názov cieľového atribútu
- Názov ID atribútu – to je taký stĺpec v tabuľke s dátami, ktorý nie je chápaný ako vstupný atribút, ale je použitý na jednoznačnú identifikáciu záznamu.
- Maximálna hĺbka stromu
- Veľkosť trénovacej množiny v %
- Veľkosť testovacej množiny v %

Príklad volania procedúry vyzerá nasledovne:

```
execute sp_c45Main 'data_sample_c45','hypoteka','id',5,80,20
```

Prítomnosť parametra určujúceho maximálnu povolenú hĺbku stromu je nevyhnutná. Algoritmus C4.5 totiž v prípade spojitého atribútu hľadá najlepší bod delenia. Keby neexistovala aj ukončovacia podmienka v podobe maximálnej hĺbky stromu, mohla by byť hĺbka stromu v najhoršom prípade ekvivalentná súčtu počtov unikátnych hodnôt pre všetky spojité atribúty. Taký strom by nebol veľmi užitočný.



obr. 38 – Zobrazenie rozhodovacieho stromu pre algoritmus C4.5

Po prebehnutí procedúry sa zobrazí záložka *Results* s výsledkami, ktorej podobu môžeme vidieť na obr. 38. Záložka ukáže získaný rozhodovací strom ako aj skóre tohto modelu a celkový čas procesu modelovania.

K-najbližších susedov (KNN)

Algoritmus k-najbližších susedov je popísaný v teoretickom úvode tejto diplomovej práce na strane 38. V aplikácii sú implementované 2 verzie. Jedna má podobu databázovej procedúry a druhá je implementovaná priamo v aplikácii v jazyku C#.

Keď sa užívateľ rozhodne pre prvú možnosť, zobrazí sa množina ovládacích prvkov slúžiacich k nastaveniu parametrov algoritmu. Hlavným parametrom je k – počet susedov. Pod ním sa nachádzajú 3 zaškrtávacie políčka. Prvé určuje, či je cieľový atribút diskrétny (klasifikácia) alebo spojitý (predikcia). Druhé dáva možnosť užívateľovi zvoliť použitie normalizácie atribútov podľa smerodajnej odchýlky (*zScore*). V opačnom prípade sa použije

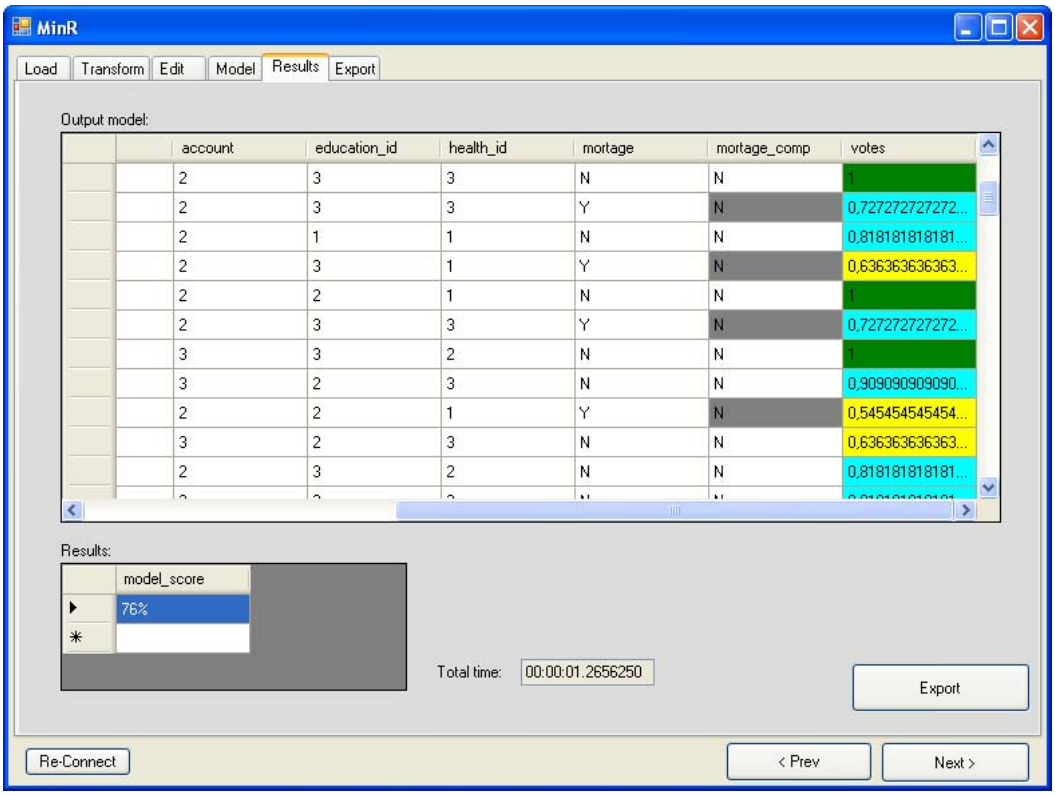
normalizácia *Min-Max*. Tretie umožňuje užívateľovi zvoliť, či hlasovanie má byť vážené alebo nie.

Tieto parametre sú spolu s názvom tabuľky, názvom cieľového atribútu, názvom ID atribútu a percentuálnou veľkosťou testovacej a trénovacej množiny predané databázovej procedúre ***sp_knnMain***.

Táto procedúra normalizuje a rozdelí dáta na testovacie a trénovacie. Následne spočíta vzdialenosti medzi jednotlivými príkladmi. Potom prebieha samotná klasifikácia/predikcia. Pri klasifikácii hlasuje *k* najbližších susedov a výsledná klasifikácia je najfrekventovanejšia z nich. Použitím váženého hlasovania (váha hlasu je nepriamo úmerná vzdialenosti suseda) možno dosiahnuť lepšie výsledky.

Pri predikcii sa volí cieľový atribút ako priemer cieľových atribútov *k* najbližších susedov. Opäť je možné použiť vážený variant, ktorý aplikuje vážený priemer. Volanie procedúry aplikáciou vyzerá nasledovne:

```
execute sp_knnMain 'data_sample_knn_cont','mort_index','id', 3,
1, 0, 0, 70, 30
```



obr. 39 – Zobrazenie výsledkov algoritmu KNN

Po úspešnom prebehnutí procedúry sa užívateľovi zobrazia jednotlivé príklady. V prípade kategoriálneho cieľového atribútu sa u každého príkladu objaví farebne označená miera jednoznačnosti hlasovania. Zelená znamená, že hlasovanie bolo úplne jednoznačné. Ukážka záložky Results pre databázovú verziu algoritmu k-najbližších susedov môžeme vidieť na obr. 39.

Aplikačné KNN

Aplikačná verzia algoritmu k-najbližších susedov bola vytvorená z cieľom porovnať rýchlosť aplikačného a databázového variantu implementácie algoritmu pre modelovanie. Tento test je detailne popísaný na strane 48.

Jediným parametrom algoritmu je voľba počtu hlasujúcich susedov. Aplikačná verzia nepodporuje spojitý cieľový atribút (predikciu), vážené hlasovanie ani neumožňuje voľbu normalizácie. Výsledné okno pre aplikačné KNN je identické s tým pre databázovú verziu, ktoré môžeme vidieť na obr. 39.

Testovanie algoritmov

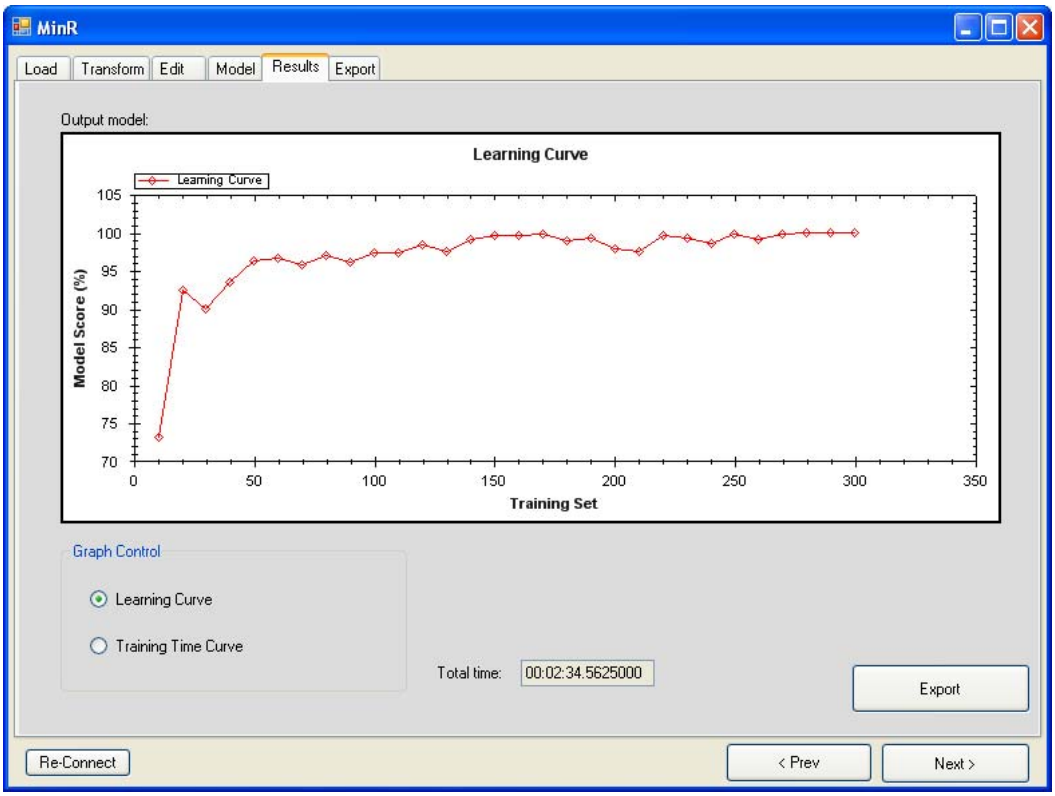
V prípade, že sa užívateľ rozhodne testovať niektorý z klasifikačných algoritmov a zvolí spustenie viacerých behov (zaškrtávacie pole *Occuring*), spustí sa procedúra ***sp_runClassTest***. Procedúra má nasledujúce parametre:

- Názov procedúry pre zvolený algoritmus
- Názov tabuľky, ktorá obsahuje dáta
- Reťazec obsahujúci špecifické parametre algoritmu
- Veľkosť trénovacej množiny v %
- Veľkosť testovacej množiny v %
- Veľkosť kroku (v percentách), o ktorý sa bude veľkosť trénovacej množiny zmenšovať medzi jednotlivými behmi
- Počet spustení algoritmu v každom behu

Príklad volania tejto procedúry vyzerá nasledovne:

```
execute [mort].[dbo].[sp_runClassTest] 'sp_knnMain',
'data_sample_knn_cont', '''mort_index'',''id'',3,0,0,0', 60,
40, 2, 5
```

Procedúra postupne znižuje veľkosť trénovacej množiny podľa zadaného kroku. Pre každú veľkosť spustí algoritmus toľkokrát, koľko bolo zadané posledným parametrom. Pred začiatkom každého spustenia zaznamená aktuálny čas. Po skončení uloží záznam obsahujúci veľkosť trénovacej množiny, skóre a dĺžku behu do tabuľky *wrkLearnCurveData*. Táto tabuľka bude použitá na generovanie krivky učenia.

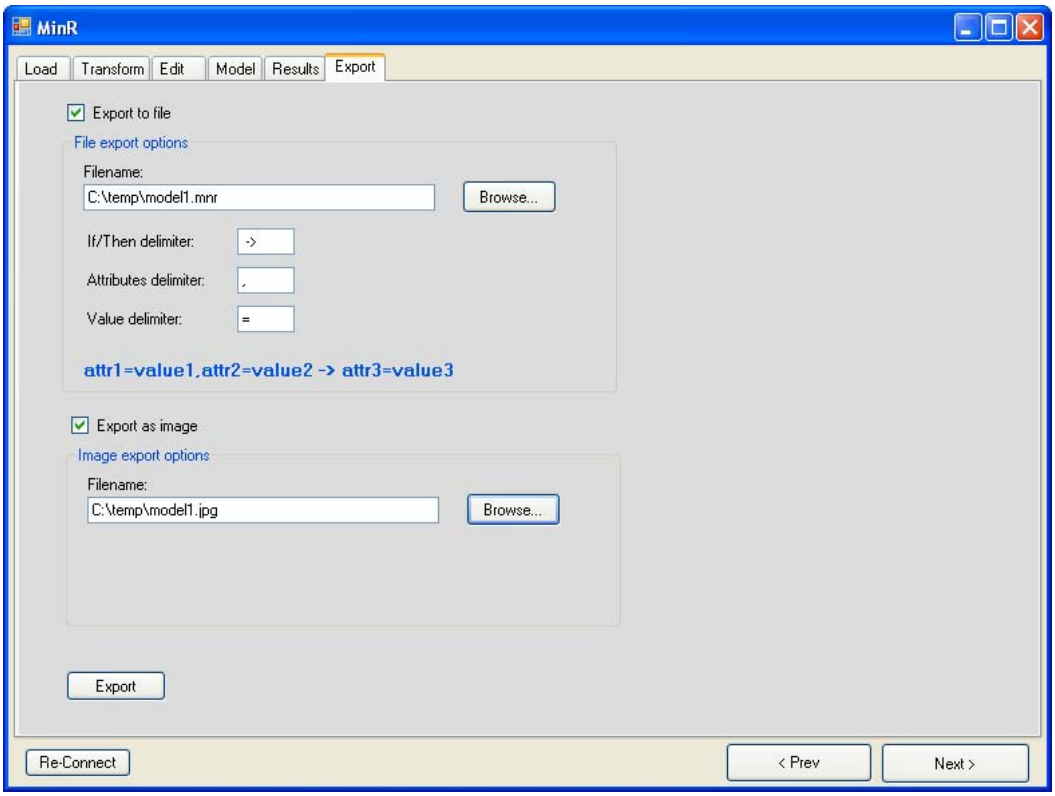


obr. 40 – Zobrazenie výsledkov testovania algoritmu ID3

Na záložke zobrazujúcej výsledky modelovania sa miesto modelu zobrazí učiaca krivka. Použitím prepínača je možné prepínať medzi učiacou krivkou (voľba *Learning Curve*) a krivkou závislosti dĺžky trénovania na veľkosti trénovacej množiny (voľba *Training Time Curve*). Na vykreslenie grafov je použitá skupina C# knižníc *ZedGraph*. Pravým tlačidlom myši možno tento graf jednoducho uložiť ako obrázok.

7.7. Export modelu

Po kliknutí na tlačidlo *Export* záložky *Results* sa objaví záložka *Export*. Užívateľ môže na tejto záložke exportovať model do textového súboru alebo ako obrázok.



obr. 41 – Exportovanie modelu

Export do súboru

Export do textového súboru je možný vykonať pre všetky algoritmy, ktoré generujú pravidlá. S výnimkou obidvoch verzií algoritmu k-najbližších susedov je teda táto možnosť prístupná pre všetky modely.

Pre export do súboru treba zaškrtnúť príslušné zaškrtávacie pole. Pod ním sa nachádza textové pole, ktoré obsahuje cestu k súboru, do ktorého budú pravidlá exportované. Tlačidlom *Browse* sa zobrazí dialógové okno komponenty *SaveFileDialog*, ktorá umožní užívateľovi vybrať súbor, kde budú pravidlá exportované. Nasledujú textové polia, ktoré umožňujú užívateľovi zvoliť oddeľovač pre strany pravidla, oddeľovač atribútov a oddeľovač názvu atribútu od jeho hodnoty. Pod ním sa nachádza reťazec, ktorý ukazuje, ako pravidlo vyzerá s aktuálne zvolenými oddeľovačmi. Po kliknutí na tlačidlo *Export* sú pravidlá zapísané do zvoleného súboru.

Export ako obrázok

Export vo forme obrázku je podporovaný iba pre rozhodovacie stromy – algoritmy ID3 a C4.5. Užívateľ opäť pomocou dialógového okna zvolí súbor a formát obrázku. Podporovaný je bitmapový formát *BMP* a komprimovaný formát *JPG*. Po kliknutí na *Export* aplikácia exportuje komponent *TreeView* vo forme obrázku.

7.8. Inštalácia aplikácie

Inštalácia databázového serveru

Aplikácia používa ako pracovnú databázu inštanciu databázového systému *Microsoft SQL Server 2005*. V prípade, že užívateľ už má tento databázový systém nainštalovaný, môže prejsť na krok inštalácie pracovnej databázy.

Inštalácia databázového systému sa vyvolá spustením inštalačného súboru „*sql_serv2005ex.exe*“, ktorý sa nachádza na priloženom CD v adresári *install*. Dôležité je pri inštalácii nezabudnúť na povolenie zmiešaného módu prihlasovania (*Mixed mode*). Podrobný návod na inštaláciu sa nachádza na adrese:

http://msdn.microsoft.com/en-us/library/ms143516(SQL.90).aspx.

Inštalácia pracovnej databázy

Vytvorenie pracovnej databázy vrátane potrebných tabuliek a uložených procedúr prebieha spustením skriptu „*sql_create.sql*“. Súbor sa nachádza v adresári *scripts* na priloženom CD. Skript možno spustiť napríklad pomocou SQL *Server Management Studia*, ktorý sa nainštaluje spolu s databázovým systémom. Dôležité je, aby užívateľ, ktorý spúšťa skript mal potrebné práva na vytváranie databáz. Preto je najlepšie použiť administrátorský účet „sa“, ktorého heslo bolo zvolené pri inštalácii.

Názov pracovnej databázy, ktorá bude vytvorená skriptom, je „*minr_db*“. V prípade potreby je možné v skripte tento názov zmeniť. Skript tiež vytvorí užívateľa *minr_user* s heslom „*minr123*“. Pri spúšťaní aplikácie je možné sa prihlasovať aj pomocou iného užívateľa, ktorý má administrátorské práva (*sysadmin*) a plné práva k pracovnej databáze.

Testovacie dáta

Testovacie dáta sa importujú spustením skriptu „*sql_data.sql*“ v databáze, ktorú si užívateľ zvolí ako zdroj svojich dát. Môže to byť aj pracovná databáza, ale tento prístup sa neodporúča. Skript sa nachádza v adresári „*scripts*“ na priloženom CD. Spustením skriptu sa vytvoria nasledovné tabuľky:

- ***data_analysis1*** – dáta z mäsokombinátu, ktoré boli analyzované a popísane v kapitole 5
- ***data_sample_knn*** – dáta, ktoré som použil na testovanie algoritmu k-najbližších susedov a porovnanie rýchlosti jeho databázovej a aplikačnej verzie
- ***data_sample_knn_cont*** – dáta, ktoré som použil na testovanie algoritmu k-najbližších susedov na predikciu (spojitý cieľový atribút)
- ***data_sample_c45*** – dáta, ktoré som použil na testovanie algoritmu C4.5.

- ***data_sample_c45_2*** – dáta, ktoré som použil na testovanie algoritmu C4.5 (obsahujú viac atribútov ako predošlá tabuľka)
- ***data_sample_classif*** – dáta, ktoré som použil na testovanie algoritmov ID3, klasifikačnej verzie analýzy nákupného košíka a algoritmu C4.5.

Aplikácia MinR

Aplikácia sa inštaluje skopírovaním súborov, ktoré sú obsiahnuté v adresári *minr* na priloženom CD do zvoleného adresára. Súbor *MinR.exe* slúži na spustenie aplikácie. Súbor *ZedGraph.dll* je knižnica, ktorá je použitá na vykresľovanie grafov a pre beh aplikácie je nevyhnutná. Pri prvom behu aplikácie je potrebné nastaviť pripojenie k pracovnej databáze. Tento proces je podrobne popísaný v kapitole 7.2 - Prihlásenie do aplikácie.

8. Záver

V závere by som rád zhrnul, do akej miery sa mi podarilo naplniť ciele stanovené v úvode práce. Chcel by som tiež poukázať na vlastné výsledky a prínosy tejto diplomovej práce. V neposlednom rade by som chcel popísať možnosti nadviazania na túto diplomovú prácu.

Aplikácia

Tvorba aplikácie zabrala značnú časť času stráveného tvorbou diplomovej práce. Ako šťastné sa ukázalo rozhodnutie použiť na vývoj užívateľského rozhrania vývojové prostredie *MS Visual Studio*. Jeho použitie mi pomohlo vytvoriť prehľadné a funkčné rozhranie v relatívne krátkom čase. To mi umožnilo sústrediť všetok čas vyhradený na vývoj aplikácie do implementácie a optimalizácie algoritmov.

Z dôvodov popísaných v mojej práci som sa rozhodol implementovať algoritmy v podobe uložených databázových procedúr. Testami som potvrdil, že tento prístup je pri veľkom množstve dát rýchlejší.

Prínosy

Diplomová práca popisuje a detailne vysvetľuje najpoužívanejšie algoritmy a celkový proces dobývania znalostí z dát. Podarilo sa mi tieto algoritmy implementovať, navzájom porovnať a výsledky zhrnúť. Výsledky sú podložené opakovateľnými testami na umelých dátach, ktoré sú dostupné na priloženom CD.

Pri výskume sa mi podarilo navrhnúť vylepšenú verziu algoritmu analýzy nákupného košíka. Tak isto som navrhol a implementoval modifikáciu tohto algoritmu na klasifikáciu (MBAC).

Dosiahnuté znalosti som aplikoval na 2 súbory reálnych dát. V oboch prípadoch som dosiahol výsledky, ktoré boli zadávateľom úlohy označené ako prínosné. V prípade klasifikačnej úlohy sa dokonca model vytvorený mojím algoritmom MBAC ukázal ako najlepší.

V neposlednom rade sa mi podarilo implementovať robustnú aplikáciu, ktorá umožňuje načítať dáta z viacerých zdrojov a implementuje množstvo algoritmov na predspracovanie dát a modelovanie.

Možné rozšírenie

Myšlienka implementácie algoritmov pracujúcich s rozsiahlymi dátami v podobe databázových procedúr sa ukázala ako dobrá. Napriek tomu však koeficient zrýchlenia nenaplnil moje očakávanie.

Keby sa ako databázový systém použil niektorý z výkonnejších databázových systémov, mohli by byť dosiahnuté lepšie výsledky. Databázový systém od spoločnosti Teradata je používaný ako dátový sklad, preto je optimalizovaný na operácie nad veľkými množstvami. Podľa oficiálnych stránok tejto spoločnosti tento systém môže byť pri spracovaní dát oproti aplikačnému prístupu rýchlejší až 25x. Keďže väčšinou je ako zdroj dát pre

dobývanie znalostí použitý dátový sklad a dáta sú rozsiahle, bolo by vhodné použiť ako pracovnú databázu práve tento systém.

9. Bibliografia

[1] Olivia Parr Rud (2001): Data Mining. *Computer Press, Praha*

[2] Vladimír Mlynarovič (2006): Indukcia rozhodovacích stromov. *STU Bratislava* http://www2.fiit.stuba.sk/~kapustik/ZS/Clanky0506/mlynarovic/index.html

[3] Jiawei Han, Micheline Kamber (2001): Data Mining – Concepts and Techniques. *Academic Press, San Diego, USA*

[4] Paul E. Utgoff (1989): Incremental Induction of Decision Trees. *University of Massachusetts* http://www.cs.umass.edu/~utgoff/papers/mlj-id5r.pdf

[5] Petr Berka (2003): Dobývaní znalostí z databází. *Academia*

[6] Breiman, L., Friedman, J.H., Olshen, R., and Stone, C.J. (1984): Classification and Regression Tree. *Wadsworth & Brooks/Cole Advanced Books & Software, Pacific California* http://support.spss.com/ProductsExt/SPSS/Documentation/Statistics/algorithms/14.0/TREE-CART.pdf

[7] Jonathan I. Maletic, Andrian Marcus (2000): Data Cleansing: Beyond Integrity Analysis. *University of Memphis* http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.5212